

PC973 U.S.P.T.O.  
09/776708  
02/05/01



## APPENDIX A

The following sections of code accomplish two tasks:

- I) Calculation of the topomeric conformation for a particular molecule, assuming that the molecule is referenced by a particular row of a Tripos Molecular Spreadsheet (MSS). With minor adaptations this code could be used in other molecular modeling environments, such as Cerius 2, Quanta, or Insight.
- II) Calculation of the *line slope* assuming that the biological data and one or more columns of property data are stored in a Tripos Molecular Spreadsheet (MSS). Almost any other software for manipulating data in a spreadsheet or other tabular representation could be adapted to perform similar calculations, assuming a Tanimoto function for expressing "distances" between bitsets of equal cardinality.

Both sections of code include procedures written in two languages. The first is C, familiar to all programmers, and includes both all specialized structure declarations and also brief explanations of all functions used. The second is SPL, an interpretative language available within the SYBYL molecular modeling program, whose syntax is similar to a Unix shell script. The SPL language is described fully in the volume entitled SPL Manual, found within the documentation set for SYBYL 6.2, release date July 1995. This volume includes descriptions of all "expression generators" (functions returning a value) and "macro commands" not specifically explained below.

### I. Topomeric Field Code:

A. SPL macro *CHOM!BUILD3D*. To build topomerically aligned 3D models, the third argument must have the value **ALIGN**, and the global associative array element **CHOM!Align[ALICYC]** must have the value **All\_trans**. Code to allow user adjustment of these and other 3D model-building parameters appearing in this code as other elements of **CHOM!Align[]** is not shown.

B. Under these circumstances the following SPL macro *CHOM!Alltrans* sets all torsions provided to their topomeric values.

C. To determine the atoms defining each torsion to be adjusted, *CHOM!Alltrans* invokes the expression generator **%trans\_path()**, which executes the following C subroutine *SYB\_MGEN\_CONN\_BEST*, with its associated subroutines *syb\_mgen\_conn\_att\_atoms*,

*get\_path\_mw*, *get\_path\_xyz*, and (if debugging) *ashow*. No user-adjustable values are used by this code. All non-obvious include files and a brief functional description of subroutines external to this code are provided in section III below.

D. The computation of rotatable-bond-attenuated steric (and/or electrostatic, hydrogen bonding) fields for the topomerically aligned conformation is carried out by the C subroutine

*QSAR\_FIELD\_EVAL\_RB\_ATTEN*, which uses the accompanying subroutine *QSAR\_FIELD\_RB\_WTS* to generate an attenuated weight for each atom's contribution to the field(s). (Pseudo code for the latter subroutine appears in its header comment.) The attenuation factor (recommended value of 0.85) is a user-adjustable or "tailorable" value, here shown as **COMFA!AGGREG\_SCALING**. The user-adjustable **HBOND\_RAD\_SCALING** parameter affects the steric "radius" of a hydrogen-bonding hydrogen.

## II. Patterson-Distribution Validation Code

A. The SPL expression generator *lrt\_fast* returns the slope of the "best" line along with the count of data points and the fractional area, within a "virtual" or conceptual graph of absolute differences in biological activities vs absolute differences in the diversity measurement to be validated. The format of its output appears in the header comment.

B. The short SPL expression generator *dochi* shows the computation of the chi-squared statistic resulting from the output of the *lrt\_fast* expression generator.

C. The C code functions *QSHELL\_HIER\_LRT*, *QSHELL\_HIER\_DO\_LRT*, and *fpt\_heapsort* generate the results produced by *lrt\_fast*. These routines generate the biological differences themselves but rely on some external procedure, not shown, to generate the distances between the diversity measurements. (The reason is that the method of calculating differences depends on the diversity parameter(s). Typically a Euclidean distance is calculated for scalar properties, or a Tanimoto difference is calculated for bitsets, and if multiple parameters are combined to form the diversity measurement to be validated then the relative weighting must also be specified by the user.)

Section III. Supporting information for interpretation of the C code in Sections I and II.

- A. Declarations of complex and non-standard data structures referenced by the declarations within these C procedures, specifically for molecules, atoms, and the regions, fields, and other user input information that are part of a CoMFA field description.
- B. Functional descriptions of all external subroutines called by these C procedures, ordered alphabetically.

```

# SECTION I-A. Macro BUILD_3D for generating and storing topomeric alignments
#
#=====
@macro BUILD_3D CHOM
# builds 3D models,
#   storage in a database or in a conformer column
#
# either not-aligned (just uses Concord or as-is if from Unity,
#   or minimizes input structure)
# or aligned for CoMFA (requires core structure as alignment template)
#   with optional fixup of side chains, charge calculation
# $1 is row ids in current MSS
# $2 is storage code (will retrieve structure from same place or somewhere)
# $3 is align (U or A)
# $4 is basic building technique
# other arguments, used only if ALIGN is true, are elements
#   of the global associative array CHOM!ALIGN

# set up mol retrieval from MSS to be fast and clean

localvar AFFECT_SUBSET_save
localvar EXAMINE_TAILOR_MODE_save
localvar HIGHLIGHT_MSS_save
localvar INFORM_save
localvar INPUT_MODE_save
localvar RELATE_save
localvar SHOW_MOLECULE_save
localvar USER_FUNCTION_save natmcore heavy ys
localvar align ma rid cgq_save tailor_bumps_save newc \
    a b max_save usehs rat yrat nrat noth

setvar AFFECT_SUBSET_save      $TAILOR!EXAMINE!AFFECT_SUBSET
setvar EXAMINE_TAILOR_MODE_save $TAILOR!EXAMINE!EXAMINE_TAILOR_MODE
setvar HIGHLIGHT_MSS_save      $TAILOR!EXAMINE!HIGHLIGHT_MSS
setvar INFORM_save             $TAILOR!EXAMINE!INFORM
setvar INPUT_MODE_save         $TAILOR!EXAMINE!INPUT_MODE
setvar RELATE_save             $TAILOR!EXAMINE!RELATE
setvar SHOW_MOLECULE_save     $TAILOR!EXAMINE!SHOW_MOLECULE
setvar USER_FUNCTION_save      $TAILOR!EXAMINE!USER_FUNCTION
setvar cgq_save $CGQ_TIMEOUT
set CGQ_timeout 0

setvar TAILOR!EXAMINE!AFFECT_SUBSET      NONE
setvar TAILOR!EXAMINE!EXAMINE_TAILOR_MODE SILENT
setvar TAILOR!EXAMINE!HIGHLIGHT_MSS      NO
setvar TAILOR!EXAMINE!INFORM            NO
setvar TAILOR!EXAMINE!INPUT_MODE        ROW_COLUMN_EXPR
setvar TAILOR!EXAMINE!RELATE          NO
setvar TAILOR!EXAMINE!SHOW_MOLECULE    YES
setvar TAILOR!EXAMINE!USER_FUNCTION    NONE
setvar max_save $TAILOR!MAXIMIN2!LS_STEP_SIZE $TAILOR!MAXIMIN2!MAXIMUM_ITERATION

setvar ma %table_attribute( MOL_AREA )

# if needed make new place to put output
setvar newc
switch %substr( $2 1 3 )
case NEW)
    setvar newc %math( %table( * COL COUNT ) + 1 )

```

```





```

```

setvar rat %arg( 1 %set_unpack( $sys ) )
setvar nrat %arg( 1 %atom_info( $rat NEIGHBORS ) )
setvar yrat %arg( 1 %set_unpack( %set_diff( \
    %set_create( %atom_info( $nrat NEIGHBORS ) ) $rat ) ) )
    ORIENT USER $nrat $rat $yrat >$nulldev
endif

# identify all the non-primary atoms for FIT, in/out of the search pattern
# and all the basic torsions (bonds to Ys) that potentially need setting
    setvar tpat %arg( 1 %search2d( %cat( %sln( $mcore ) ) $capsln NoDup 0 y ) )
setvar hvinpat
setvar patats
setvar tors
setvar usehs
setvar sybhvats %set_create(%atoms(*-<H>))
if %lt( %set_size( $sybhvats ) 3 )
    setvar usehs TRUE
    setvar sybhvats %set_create(%atoms(*))
endif

for a in %range(1 %sln_atom_count( $capsln ) )
    if %or( "$usehs" "%not( %set_and( %sln_atom_symbol( $capsln $a ) \
        H,F,Cl,Br,I ) )" )
# for FIT, need to know the SYBYL IDs of the heavy atoms
    setvar hvinpat $hvinpat $a
    setvar patats[ $a ] %sln_rgroup_sybid( $mcore $tpat $a )
    setvar patats[ $a ][ YS ] %set_and( "sys" "%set_create( \
        %atom_info( $patats[ $a ] NEIGHBORS ) )" )
# for each torsion root, need to save the SLN ID of an arbitrary
# heavy atom torsional definer
    if $patats[ $a ][ YS ]
        setvar tors[ $a ] %set_and( %set_diff( "%set_create( \
%atom_info( $patats[ $a ] NEIGHBORS ) )" $patats[ $a ][ YS ] ) $sybhvats )
# if there are several possibilities, prefer the lowest #'d carbon
# to define trans-ness
        if %gt( %set_size( $tors[ $a ] ) 1 )
            if %set_and( $tors[ $a ] %set_create( %atoms(<C*>) ) )
                setvar tors[ $a ] %set_and( $tors[ $a ] \
                    %set_create( %atoms(<C*>) ) )
            endif
            setvar tors[ $a ] %arg( 1 %set_unpack( $tors[ $a ] ) )
        endif
        for a1 in %range(1 %sln_atom_count( $capsln ) )
            if %eq( $tors[ $a ] %sln_rgroup_sybid( $mcore $tpat $a1 ) )
                setvar tors[$a] $a1
                break
            endif
        endfor
    endif
endif
endfor
if $CHOM!Align[DEBUG]
echo %prompt( INT 1 " " " " )
endif
endif

default $ma >$nulldev
setvar CHOM!BadRows

```

```

## build 3D models
## off we go !! Get MSS row IDs to build models for
if %streql( $1 * )
    setvar rids %table( * ROW NUM )
else
    setvar rids %set_unpack( $1 )
endif

for rid in $rids

# get the next MSS entry to be modelled
table examine $rid | | >$nulldev

# fix NO2's (egad what a pain) because Concord & SYBYL are inconsistent
setvar pat %search2d( %sln( $ma ) N(=O)O ALL 0 y )
while $pat
    setvar pat %sln_rggroup_sybid( $ma %arg( 1 $pat ) 1 3 )
    modify bond type %bonds( %cat( %arg( 1 $pat ) "=" \
        %arg( 2 $pat ) ) ) 2 >$nulldev
    modify atom type %arg( 2 $pat ) O.2
    setvar pat %search2d( %sln( $ma ) N(=O)O ALL 0 y )
endwhile

if $CHOM!Align[DEBUG]
    label id *
endif

# basic optimization
switch $4
case CONCORD)
    CONCORD MOL $ma >$nulldev
# if Concord failed, we may still be awfully flat
# minimize if there are heavy atoms not part of a single aromatic system ..
    setvar noth %atoms( *-<H> )
    setvar a1 %arg( 1 $noth )
    if %set_diff( "%set_create( $noth )" \
        "%set_create( %atoms( %cat( "aromatic( \"$a1\" )" ) ) ) ) "
        setvar zs %extent_3d( %cat( $ma "(*)" ) )
        setvar zs %math( %arg( 5 $zs ) - %arg( 6 $zs ) )
        if %eq( $zs 0.0 )
            %unflatten( %cat( $ma "(*)" ) )
            MAXIMIN $ma DONE INTERACTIVE
        endif
    endif
;;
case MINIMIZE)
    MAXIMIN $ma DONE INTERACTIVE >$nulldev
;;
endswitch

# done, if only 3d coord, but for topomeric CoMFA ..
if %streql( %substr( $3 1 1 ) "A" )
# find any arbitrary 2D hit
    setvar pat %search2d( %cat( %sln( $ma ) ) $capsln NoDup 0 y )
    if %not( $pat )
        setvar CHOM!BadRows %set_or( "$CHOM!BadRows" $rid )
        echo $capsln not found in molecule for Row $rid .. skipping
        goto next1

```

```

endif
setvar pat %arg(1 $pat )
setvar allpatats %set_create( %sln_rgroup_sybid( $ma $pat \
%range( 1 %sln_atom_count( $capsln ) ) ) )

# collect all appropriate heavy atoms for FIT and torsions
setvar mat1
setvar mat2
setvar schns
for a in $hvinp
    setvar mat1 $mat1 $patats[ $a ]
    setvar sybat %sln_rgroup_sybid( $ma $pat $a )
    setvar mat2 $mat2 $sybat
# are there heavy atom neighbors to FIT also (and generate torsion lists)?
if $patats[$a] [YS]
    setvar ans %set_diff( %set_create( \
        %atom_info( $sybat NEIGHBORS ) ) $allpatats )
    setvar ans %atoms($ans-<H>)
    setvar i 1
    for p in %set_unpack( $patats[$a] [YS] )
# add heavy atom neighbors to FIT list
    if %arg( $i $ans )
        setvar mat1 $mat1 $p
        setvar mat2 $mat2 %arg( $i $ans )
# generate another torsion for CHOM!alltrans
        setvar schns $schns %cat( $sybat "," \
%sln_rgroup_sybid( $ma $pat $tors[ $a ] ) "," %arg( $i $ans ) )
    endif
    setvar i %math( $i + 1 )
endfor
endif
endfor

setvar dofit MATCH %cat( $mcore "(" %set_create( $mat1 ) ")" ) \
%cat( $ma "(" %set_create( $mat2 ) ")" )
$doFit >$nulldev
if $CHOM!Align[DEBUG]
echo %prompt( INT 1 " " " " )
endif

# do FIT
if %gt( $MATCH_RMS $CHOM!Align[ FITRMS ] )
    setvar CHOM!BadRows %set_or( "$CHOM!BadRows" $rid )
    echo Bad geometric alignment (MATCH_RMS = $MATCH_RMS) for Row $rid .. sk
    goto next1
endif
# side chain alignments ..
switch $CHOM!Align[ ALICYC ]
case User_Macro)
    $CHOM!Align[ ALIDATA ] $ma $CHOM!ALIGN[ MCORE ]
;;
case All_trans)
case With_Templates)
    setvar nojrrings TRUE
    setvar rbds %set_create( %bonds({rings()}))
    for i in $schns
        setvar jbd %set_unpack( $i )
# can set "side chain" bonds only if connecting bond is not cyclic
        if %set_and( "$rbds" "%bonds( %cat( %arg( 3 $jbd ) = \

```

```

        %arg( 1 $jbds ) ) ) )
    setvar nojrings
else
    CHOM!AllTrans $jbds
endif
endfor
if $CHOM!Align[DEBUG]
    echo %prompt( INT 1 " " " " )
endif
if %streq( $CHOM!Align[ ALICYC ] With_Templates )
    setvar f %open( $CHOM!Align[ ALIDATA ] "r" )
    setvar buff %read( $f )
    setvar slnma %cat( %sln( $ma ) )
    while $buff
# each line of text should have pattern, SLN IDs for the 4 torsion atoms,
# and a torsion value to set
        if %eq( %count( $buff ) 5 )
            setvar torpat %search2d( $slnma %arg( 1 $buff ) NoDup 0 y )
            for t in $torpat
                MODIFY TORSION %sln_rgroup_sybid( $ma $t %arg( 2 $buff ) \
%arg( 3 $buff ) %arg( 4 $buff ) ) %arg( 5 $buff ) >$nulldev
            endfor
        endif
    endwhile
    %close( $f )
endif
;;
endswitch
endif

# do a bump check?
if $CHOM!Align[BUMPS]
    %if %atoms({bumps(*,*)})
        setvar CHOM!BadRows %set_or( "$CHOM!BadRows" $rid )
        echo Bad steric contacts in aligned conformer for Row $rid .. skipping
        goto nextl
    %endif
endif
# partial charges ..
switch $CHOM!Align[ CHARGE ]
case None)
;;
case User Macro)
    exec $CHOM!Align[ CHARGEDATA ] $ma
;;
case )
    CHARGE $ma COMPUTE $CHOM!Align[ CHARGE ] | >$nulldev
;;
endswitch

# put conformer away
switch %substr( $2 1 3 )
case SYB)
    database add $ma r >$nulldev
;;
case )
    %wcell( $rid $newc %cat( %cat( %sln( $ma FULL CHARGE ) ) ) ) >$nulldev
;;
endswitch

```

```

echo Built row $rid
next1:
endfor

if %streq( %substr( $3 1 1 ) "A" )
  copy $mcsav $mcore
  zap $mcsav
endif

if $CHOM!Align[BUMPS]
  TAILOR SET GENERAL bumps_contact_distance $tailor_bumps_save | |
endif
# done, restore initial EXAMINE settings

set CGQ_TIMEOUT $cgq_save
setvar TAILOR!EXAMINE!AFFECT_SUBSET $AFFECT_SUBSET_save
setvar TAILOR!EXAMINE!EXAMINE_TAILOR_MODE $EXAMINE_TAILOR_MODE_save
setvar TAILOR!EXAMINE!HIGHLIGHT_MSS $HIGHLIGHT_MSS_save
setvar TAILOR!EXAMINE!INFORM $INFORM_save
setvar TAILOR!EXAMINE!INPUT_MODE $INPUT_MODE_save
setvar TAILOR!EXAMINE!RELATE $RELATE_save
setvar TAILOR!EXAMINE!SHOW_MOLECULE $SHOW_MOLECULE_save
setvar TAILOR!EXAMINE!USER_FUNCTION $USER_FUNCTION_save
TAILOR SET MAXIMIN2 LS_STEP_SIZE %arg( 1 $max_save ) \
  MAXIMUM_ITERATIONS %arg( 2 $max_save ) | |

# update row and column information

if %streq( %substr( $2 1 3 ) NEW )
# make any new conformer column become the source of molecules
  TABLE CONF %table( * COL COUNT )
  CHOM!UPDATE_ROW_SEL $CHOM!CID_Last
  setvar CHOM!CID_Last %math( $CHOM!CID_Last + 1 )
else
  CHOM!UPDATE_ROW_SEL
endif

#
# Section I-B. Generates the topomeric conformation of the 3D model
#
=====

@macro ALLTRANS chom
# assumes default molecule, takes argument atoms $1 and $2
# where $1 is the JOINed atom of the core, $2 is the atom that
#   the rest of the substituent is to be trans to,
#   and $3 is the JOINed atom of the substituent
# starts from that atom and sets all side chains
#   to a topomeric conformation

localvar bds b bdset a1 a2 tmp sbonds sats rbond pbds torsion ringbonds doit

# check input for legality
  setvar tmp %set_create( %atom_info( $1 NEIGHBORS ) )
  if %not( %eq( 2 %count( %set_unpack( %set_and( \
    "$tmp" %cat( $2 "," $3 ) ) ) ) ) )
    echo Bad input to ALLTRANS (atoms $2 $3 not bonded to $1)
    return

```

```

endif

# save key bonds
setvar rbond %bonds( %cat( $3 "=" $1 ) )
setvar sats %conn_atoms( $3 $1 )
if %not( $sats )
#    echo No substituent atoms found in ALLTRANS
    return
endif

setvar sats $3 $sats
setvar sbonds %set_create( %bonds( \
    %cat( "{TO_ATOMS(" %set_create($sats) ")" }" ) ) )

# define the other bonds that might need adjusting
setvar bds %set_create( %bonds( {*-{RINGS()}}&<1> ) )
setvar bds %set_and( "$sbonds" "$bds" )
if %not( $bds )
    return
endif

# discard bonds to primary atoms
setvar mval %set_create( %atoms( \
    <H>+<O.2>+<F>+<I>+<Cl>+<Br>+<n.1>+<LP>+<Du> ) )
setvar pds %set_create( %bonds( %cat( "{TO_ATOMS(" $mval ")" }" ) ) )
setvar bds %set_diff( $bds $pds )
setvar ringbonds %set_create(%bonds({RINGS()}))

# walk all the important bonds
for b in %set_unpack( $bds )
    setvar doit TRUE
# if this is the JOIN bond, already have some info
    if %eq( $b $rbond )
        setvar a0 $2
        setvar a1 $1
        setvar a2 $3
# still need to be SURE we're not monovalent
    if %or( "%eq( 1 %count( %atom_info( $a1 NEIGHBORS ) ) )" \
        "%eq( 1 %count( %atom_info( $a2 NEIGHBORS ) ) )" )
        setvar doit
    endif
    else
        setvar bdat %bond_info( $b ORIGIN TARGET )
        setvar a1 %arg( 1 $bdat )
        setvar a2 %arg( 2 $bdat )
        if %or( "%eq( 1 %count( %atom_info( $a1 NEIGHBORS ) ) )" \
            "%eq( 1 %count( %atom_info( $a2 NEIGHBORS ) ) )" )
            setvar doit
        endif
        if $doit
# which end leads to root atom? if necessary flip a1,a2 to make that one be a1
        if %set_and( "%set_create( %conn_atoms( $a2 $a1 ) )" $1 )
            setvar tmp $a1
            setvar a1 $a2
            setvar a2 $tmp
        endif
        setvar a0 %trans_path( $a1 $a2 $1 )
    endif
endif
if $doit
    setvar a3 %trans_path( $a2 $a1 )

```

```

switch %count( %unpack( "%set_and( "$ringbonds" \
    %set_create( %bonds( %cat( $a0 "=" $a1 "," $a2 "=" $a3 ) ) ) ) ) )
case 0)
    setvar torsion 180
;;
case 1)
    setvar torsion 90
;;
case 2)
    setvar torsion 60
;;
endswitch
modify torsion $a0 $a1 $a2 $a3 $torsion >$nulldev
endif
endfor

```

/\* Beginning of section I-C, C code implementing the trans\_path expression gener

```

/*E+:SYB_MGEN_CONN_BEST*/
/*********************************************************************
* int SYB_MGEN_CONN_BEST( identifier, nargs, args, writer )           *
*   Dick Cramer, Apr. 9, 1995 (written for SELECTOR use)             *
*   @ Expression generator that returns the atoms attached to a given   *
*     atom, excepting the second, in a prioritized order.                 *
*   If there are two arguments, the ordering is by decreasing branch      *
*   @ "size", where "size" is first any path with rings encountered, then*
*     number of attached atoms, then MW (paths in cycles end when an atom*
*     in another path is encountered.)                                     *
*   @ If three arguments, the atom that is returned is the one that       *
*     begins the shortest path containing the atom referred to by the     *
*     third argument. If multiple such paths, ordering is same as for      *
*     two arguments.                                                       *
*   @ Further prioritization of paths is by molecular weight,            *
*     and then by lowest X, Y, Z values.                                    *
*   @ If last argument is DEBUG, all paths are written to stdout.          *
*   @ User interface:                                                     *
*     %trans_path( a1 a2 ( a3 ) (DEBUG) )                                */
*****
```

```

int SYB_MGEN_CONN_BEST( identifier, nargs, args, Writer )
/* following arguments contain the text supplied to the %trans_path()
   expression generator, and provide an avenue for producing text output. */
char *identifier;
int nargs;
char *args[];
PFI Writer;
{
```

```
# define MAX_NP 8
```

```

    struct pathrec {
        int root, nrings, chosen, nats;
        float mw, xyz[3];
        set_ptr path;
    } ;
```

```
    struct pathrec p[MAX_NP];
```

```

int retval, i, np, toroot, a1, a2, a4, a, pnow, pdone, growing,
    final_pos, area_num, new_rings, nats, nuats, elem, ncycles,
    best, debug, ringclosed;
List_Ptr      atom_exp_list=NIL,SYB_EXPR_ANALYZE();
mol_ptr       m1, m2, SYB_AREA_GET_MOLECULE();
atom_ptr      arec, SYB_ATOM_FIND_REC();
/* A set_ptr data structure is a Boolean set, first word containing
its cardinality. */
set_ptr      atom_set1=NIL, a2chk = NIL, nuls = NIL, cnats = NIL,
            nxcn = NIL, end_atoms = NIL, scratch = NIL,
            SYB_ATOM_FIND_SET(), UTL_SET_CREATE();
char         tempString[256];
float        get_path_mw(), diff;
void         get_path_xyz();

retval = 0;
/* Check the number of arguments */
if ( nargs < 2 || nargs > 4 ) {
    UIMS2_WRITE_ERROR(
        "Error: %trans_path requires 2 to 4 arguments\n" );
    return 0;
}
np = 0;
debug = (!UTL_STR_CMP_NOCASE( args[ nargs - 1 ], "DEBUG" ));
toroot = (debug && nargs == 4) || (!debug && nargs == 3);

/* PARSE THE INPUT */

/* get first atom */
if (!(atom_exp_list = SYB_EXPR_ANALYZE( SYB_EXPR_GET_ATOM_TOKEN, args[0],
    &final_pos, &area_num )))
    goto error;

if (!(m1 = SYB_AREA_GET_MOLECULE (area_num)))
    goto cleanup;
if (!(atom_set1 = SYB_ATOM_FIND_SET ( m1, atom_exp_list)))
    goto error;
if( atom_exp_list)
    SYB_EXPR_DELETE_RPN_LIST( atom_exp_list);
atom_exp_list = (List_Ptr) NIL;

if(!(1 == UTL_SET_CARDINALITY(atom_set1))) {
    UIMS2_WRITE_ERROR(
        "Error: First argument must be only one atom\n" );
    goto error;
}
if (!(arec = SYB_ATOM_FIND_REC (m1, UTL_SET_NEXT (atom_set1, -1)) )) goto er
a1 = arec->recno;
UTL_SET_DESTROY( atom_set1 );
atom_set1 = NIL;
/* get 2nd atom */
if (!(atom_exp_list = SYB_EXPR_ANALYZE( SYB_EXPR_GET_ATOM_TOKEN, args[1],
    &final_pos, &area_num )))
    goto error;

if (!(m2 = SYB_AREA_GET_MOLECULE (area_num)))
    goto cleanup;
if (!(end_atoms = SYB_ATOM_FIND_SET ( m2, atom_exp_list)))
    goto error;

```

```

if( atom_exp_list)
    SYB_EXPR_DELETE_RPN_LIST( atom_exp_list);

atom_exp_list = (List_Ptr) NIL;

if (m1 != m2 ) {
    UIMS2_WRITE_ERROR(
        "Error: atoms must be in the same molecule\n");
    goto error;
}

if (!(1 == UTL_SET_CARDINALITY(end_atoms))) {
    UIMS2_WRITE_ERROR(
        "Error: Second argument must be only one atom\n");
    goto error;
}
if (!(arec = SYB_ATOM_FIND_REC (m1, UTL_SET_NEXT (end_atoms, -1)) )) goto er
a2 = arec->recno;

/* get 3rd atom */
if (toroot) {
    if (!(atom_exp_list = SYB_EXPR_ANALYZE( SYB_EXPR_GET_ATOM_TOKEN, args[2],
        &final_pos, &area_num)))
        goto error;

    if (!(m2 = SYB_AREA_GET_MOLECULE (area_num)))
        goto cleanup;
    if (!(atom_set1 = SYB_ATOM_FIND_SET ( m2, atom_exp_list)))
        goto error;
    if( atom_exp_list)
        SYB_EXPR_DELETE_RPN_LIST( atom_exp_list);

    atom_exp_list = (List_Ptr) NIL;

    if (m1 != m2 ) {
        UIMS2_WRITE_ERROR(
            "Error: atoms must be in the same molecule\n");
        goto error;
    }

    if (!(1 == UTL_SET_CARDINALITY(atom_set1))) {
        UIMS2_WRITE_ERROR(
            "Error: Second argument must be only one atom\n");
        goto error;
    }
    if (!(arec = SYB_ATOM_FIND_REC (m1, UTL_SET_NEXT (atom_set1, -1)) )) goto er
    a4 = arec->recno;

    UTL_SET_DESTROY( atom_set1 );
    atom_set1 = NIL;
}
/* GENERATE the paths */

/* set up paths */
if (!(a2chk = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;
if (!(nuls = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;
if (!(cnats = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;
if (!(nxcn = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;
if (!(scratch = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;

if (!syb_mgen_conn_att_atoms( a2chk, m1, a1 )) goto error;
if (!UTL_SET_MEMBER( a2chk, a2 )) {

```

```

UIMS2_WRITE_ERROR (
    "Error: second argument atom is not bonded to first argument atom/\n")
    goto error;
}

UTL_SET_DELETE( a2chk, a2 );
a = -1;
np = 0;
while (np < MAX_NP && (a = UTL_SET_NEXT( a2chk, a )) >= 0 ) {
    if (! (p[np].path = UTL_SET_CREATE( m1->max_atoms + 1 ) )) goto error;
    p[np].root = a;
    p[np].nrings = 0;
    UTL_SET_INSERT( p[np].path, a );
    np++;
}

/* grow the paths */
growing = TRUE;
nats = 0;
ncycles = 0;
while (growing) {
    nuats = 0;
    ringclosed = FALSE;
    for (pnow = 0; pnow < np; pnow++) {
        UTL_SET_COPY_INPLACE( cnats, p[pnow].path );
        UTL_SET_CLEAR( nxcn );
        elem = -1;
/* accumulate this generation of attached atoms into nxcn */
        while ( (elem = UTL_SET_NEXT( cnats, elem)) >= 0 ) {
            UTL_SET_CLEAR( nuls );
            if (!syb_mgen_conn_att_atoms( nuls, m1, elem )) return( FALSE );
            UTL_SET_DELETE( nuls, a1 );
            UTL_SET_DIFF_INPLACE( nuls, end_atoms, nuls );

            UTL_SET_OR_INPLACE( nxcn, nuls, nxcn );
            UTL_SET_DIFF_INPLACE( nxcn, p[pnow].path, nxcn );
        }
        UTL_SET_OR_INPLACE( p[pnow].path, nxcn, p[pnow].path );
/* remove and mark ring closures when growing out */
        if (!toroot) for (pdone = 0; pdone < np; pdone++) if (pdone != pnow) {
            UTL_SET_AND_INPLACE( p[pnow].path, p[pdone].path, a2chk );
            if !(new_rings = UTL_SET_CARDINALITY( a2chk ))) {
/* we have ring closure(s) */
                p[pnow].nrings += new_rings;
                p[pdone].nrings += new_rings;
                ringclosed = TRUE;
                UTL_SET_OR_INPLACE( end_atoms, a2chk, end_atoms );
/* if pdone < pnow, two branches are now same lengths, drop common atom from bot
   but if >, branches are different, and must avoid repeated closing */
                if (pdone < pnow) {
/* remove atom(s) in the previous branch because paths are really same length
                    UTL_SET_DIFF_INPLACE( p[pdone].path, a2chk, p[pdone].path );
                    UTL_SET_DIFF_INPLACE( p[pnow].path, a2chk, p[pnow].path );
                }
                else {
/* must identify and mark each atom in nxcn that is attached to a2chk atom */
                    elem = -1;
                    while ( (elem = UTL_SET_NEXT( a2chk, elem)) >= 0 ) {
                        UTL_SET_CLEAR( scratch );
                        if !(syb_mgen_conn_att_atoms( scratch, m1, elem ))
                            return( FALSE );
                        UTL_SET_AND_INPLACE( scratch, nxcn, scratch );
                    }
                }
            }
        }
    }
}

```

```

        UTL_SET_OR_INPLACE( end_atoms, scratch, end_atoms );
    }
}

/* done growing paths if no more atoms added to any path .. */
for (pdone = 0, nuats = 0; pdone < np; pdone++ )
    nuats += UTL_SET_CARDINALITY( p[pdone].path );
if (nuats<=nats && !ringclosed) growing = FALSE;
nats = nuats;
/* .. or looking for the 4th atom and found it .. */
if (toroot) for (pdone = 0; pdone < np; pdone++ )
    if (UTL_SET_MEMBER( p[pdone].path, a4 )) growing = FALSE;
/* .. or after 100 atom layers out regardless */
ncycles++;
if (ncycles >= 100) growing = FALSE;
}
/* debugging */
if (debug) for (pdone = 0; pdone < np; pdone++) {
    sprintf( tempString, "Path %d (%d rings, from %d): ",
        pdone+1, p[pdone].nrings, p[pdone].root );
    UBS_OUTPUT_MESSAGE( stdout, tempString );
    ashow( p[pdone].path, m1 );
}
/* compute the path properties */
for (pdone = 0; pdone < np; pdone++) {
/* mark as already chosen any path that can't be an answer */
    p[pdone].chosen = toroot && !UTL_SET_MEMBER(p[pdone].path, a4);
    p[pdone].nats = UTL_SET_CARDINALITY( p[pdone].path );
    p[pdone].nrings = p[pdone].nrings ? 1 : 0;
    p[pdone].mw = 0.0;
    p[pdone].xyz[0] = p[pdone].xyz[1] = p[pdone].xyz[2] = 0.0;
}
/* return the best result */
best = 0;
for (pdone = 1; pdone < np; pdone++) {
    if (toroot) {
        if (p[best].chosen && !p[pdone].chosen) best = pdone;
/* looking backward along chain, always grow away from more negative coord value
        if (!p[best].chosen && !p[pdone].chosen) {
            get_path_xyz( p[pdone].root, m1, p[pdone].xyz );
            get_path_xyz( p[best].root, m1, p[best].xyz );
            for ( i = 0; i < 3; i++ ) {
                diff = p[pdone].xyz[i] - p[best].xyz[i];
                if (diff < -0.1) {
                    best = pdone;
                    break;
                }
                if (diff > 0.1 ) break;
/* checking other coords if basically tied at this coord */
            }
        }
    }
    else {
        if (p[pdone].nrings && !p[best].nrings) best = pdone;
        else if (p[pdone].nats > p[best].nats) best = pdone;
        else if (p[pdone].nats == p[best].nats) {
            p[pdone].mw = get_path_mw( p[pdone].path, m1, p[pdone].mw );
            p[best].mw = get_path_mw( p[best].path, m1, p[best].mw );
        }
    }
}

```

```

        if (p[pdone].mw > p[best].mw) best = pdone;
    }
}
arec = SYB_ATOM_FIND_REC( m1, p[best].root );
sprintf(tempString, "%d", arec->id );
if(!(*Writer)(tempString)) goto error;

retval = TRUE;
error:
cleanup:
if( atom_exp_list)
    SYB_EXPR_DELETE_RPN_LIST( atom_exp_list);
if(atom_set1)
    UTL_SET_DESTROY(atom_set1);
if(end_atoms)
    UTL_SET_DESTROY(end_atoms);
if(a2chk)
    UTL_SET_DESTROY(a2chk);
if(nuls)
    UTL_SET_DESTROY(nuls);
if(nxcn)
    UTL_SET_DESTROY(nxcn);
if(cnats)
    UTL_SET_DESTROY(cnats);
if(scratch)
    UTL_SET_DESTROY(scratch);
}
return( retval );
}

static int syb_mgen_conn_att_atoms( aset, m, atid )
/* adds atoms attached to atm into aset */
/* WORKS STRUCTLY WITH RECNOS */
set_ptr aset;
mol_ptr m;
int atid;
{
atom_ptr at, SYB_ATOM_FIND_ID();
List_Ptr tohs, UTL_LIST RETRIEVE_P();
atom_ptr toh, SYB_ATOM_FIND_REC();
acon_ptr conn1;
int nbytes1;

at = SYB_ATOM_FIND_REC( m, atid );
tohs = at->conn_atom;
while (tohs) {
    tohs = UTL_LIST RETRIEVE_P( tohs, &conn1, &nbytes1);
    toh = SYB_ATOM_FIND_REC( m, conn1->target );
    UTL_SET_INSERT( aset, toh->recno );
}
return( TRUE );
}

static float get_path_mw( aset, m, mw )
/* returns the total atomic weight of all atoms in aset */
set_ptr aset;
mol_ptr m;
float mw;

```

```

{
    int elem = -1;
    float ans = 0.0;
    atom_ptr at, SYB_ATOM_FIND_REC();
    fpt SYB_ATAB_ATOMIC_WEIGHT();

    if (mw) return( mw );
    elem = -1;
    while ( (elem = UTL_SET_NEXT( aset, elem)) >= 0 ) {
        at = SYB_ATOM_FIND_REC( m, elem );
        ans += (float) SYB_ATAB_ATOMIC_WEIGHT( at->type );
    }
    return( ans );
}

static void get_path_xyz( aid, m, mw )
/* returns the xyz of the supplied atom */
int aid;
mol_ptr m;
float mw[3];
{
    int i;
    atom_ptr at, SYB_ATOM_FIND_REC();
    if(mw[0]) return;
    at = SYB_ATOM_FIND_REC( m, aid );
    for (i = 0; i < 3; i++) mw[i] = at->xyz[i];
    return;
}

static int ashow( aset, m )
/* for interactive debugging, shows a set's membership in terms of atom ID */
set_ptr aset;
mol_ptr m;
{
    char buff[1000], *b;
    atom_ptr at, SYB_ATOM_FIND_REC();
    int elem;

    *buff = '/0';
    b = buff;
    elem = -1;
    while ( (elem = UTL_SET_NEXT( aset, elem)) >= 0 ) {
        at = SYB_ATOM_FIND_REC( m, elem );
        sprintf( b, " %d", at->id );
        b = buff + strlen( buff );
    }
    sprintf( b, "\n" );
    UBS_OUTPUT_MESSAGE( stdout, buff );
}

/* BEGINNING OF SUBROUTINES I-D. Calculation of attenuated fields */

/*+E:QSAR_FIELD_EVAL_RB_ATTEN()*/
/***********************/
/*
/* int QSAR_FIELD_EVAL_RB_ATTEN( molp, stfldp, elfldp, regp, no_st, no_el, ctp )
/*
/* Dick Cramer May 13, 1995
/*

```

/\* "Standard CoMFA" -- except that the contribution of any atom to the field falls off with an inverse power of its distance from a root atom, measured in NUMBER OF ROTATABLE BONDS!

This means also that each individual atom's contribution has a similarly scaled upper bound, rather than checking the upper bound only for the sum over all atoms.

/\* This procedure computes vdw 6-12 steric values at each point in region \*/

/\* and the electrostatic interactions (initially assuming 1/r dielectric). \*/

/\* \*/

/\* NOTE:: initially ignoring space averaging, other user knobs. \*/

/\* note:: assuming valid input here; error checking higher up ! \*/

/\* \*/

/\* \*/

/\* Input:

/\* molp - molecule pointer, molecule to place in region. \*/

/\* stfldp - steric field pointer, where values will be placed. \*/

/\* elfldp - electrostatic field pointer, where values will be placed. \*/

/\* regp - region pointer, locations where values are to be evaluated. \*/

/\* no\_st - flag to skip steric evaluations \*/

/\* no\_el - flag to skip electrostatic evaluations \*/

/\* ctp - ComfaTopPtr, for dummy/lp values \*/

/\* \*/

/\* Returns 0 on failure, 1 otherwise. \*/

/\* \*/

\*\*\*\*\*

/\*+QSAR\_FIELD\_EVAL\_RB\_ATTEN()\*/

int QSAR\_FIELD\_EVAL\_RB\_ATTEN ( molp, stfldp, elfldp, regp , no\_st, no\_el, ctp)

mol\_ptr molp;

FieldPtr stfldp, elfldp;

RegionPtr regp;

int no\_st, no\_el ;

ComfaTopPtr ctp;

{

BoxPtr box;

atom\_ptr at, SYB\_ATOM\_FIND\_ID();

int pid, b, ix, iy, iz, nat, vol\_avg, repulsive ;

fpt \*steric, \*elect, SYB\_ATAB\_VDW\_RADII() ;

fpt diff, dis, dis2, x, y, z, sum\_steric, sum\_elect ;

fpt dis6, dis12 , repuls\_val, offs[9][3], atm\_stc, atm\_ele;

fpt \*charge, \*ctemp, \*coord, \*ftemp, \*wt, scale\_vol\_avg, atm\_steric, atm\_elect;

int \*atyp , \*itemp, dohbd, dohba, ishbd, retval, dielectric , off, atid;

static fpt hbond\_scal;

fpt hbond\_A, hbond\_B, \*AtWts = NIL, \*QSAR\_FIELD\_RB\_WTS();

int \*HAs, \*HDS, \*HAp, \*HDp; /\* sets would be more efficient but slower \*/

int do\_steric, do\_elect;

set\_ptr hdonor, SYB\_HBOND\_DONORS(), pset = NIL, aset = NIL;

#define Q2KC 332.0

#define MIN\_SQ\_DISTANCE 1.0e-4

/\* ^^^ any atom within 10^-2 Angstroms is hereby zapped !

this is about it:  $10^6 / 10^{-24}$  is close to overflow! \*/

ftemp = NIL; ctemp = NIL; itemp = NIL; retval = FALSE; HAs = NIL; HDS = NIL;  
hdonor = NIL;

/\* for now, make root atom the one closest to 0,0,0 \*/

for (nat = 1; nat <= molp->natoms; nat++) {

```

at = SYB_ATOM_FIND_ID( molp, nat );
dis2 = at->xyz[0] * at->xyz[0] + at->xyz[1] * at->xyz[1] +
      at->xyz[2] * at->xyz[2];
if (nat == 1 || dis2 < dis) {
    dis = dis2;
    atid = nat;
}
}

/* following is specific to topomeric fields */
if (! (AtWts = QSAR_FIELD_RB_WTS( molp, atid ) )) goto cleanup;

if (!no_el)
{dielectric = elfldp->dielectric ;
 vol_avg = elfldp->vol_avg_type;
 scale_vol_avg = elfldp->scale_vol_avg;
 repulsive = elfldp->repulsive;
 repuls_val=repexp[repulsive]; elect = elfldp -> field_value;}
if (!no_st)
{vol_avg = stfldp->vol_avg_type;
 scale_vol_avg = stfldp->scale_vol_avg;
 repulsive = stfldp->repulsive;
 repuls_val=repexp[repulsive]; steric = stfldp -> field_value;}

if (! (ftemp = (fpt *) UTL_MEM_ALLOC(3*sizeof(fpt)*molp->natoms))) goto cleanup;
if (! (ctemp = (fpt *) UTL_MEM_ALLOC( sizeof(fpt)*molp->natoms))) goto cleanup;
if (! (itemp = (int *) UTL_MEM_ALLOC( sizeof(int)*molp->natoms))) goto cleanup;
if (! (HAs = (int *) UTL_MEM_ALLOC( sizeof(int)*molp->natoms))) goto cleanup;
if (! (HDs = (int *) UTL_MEM_ALLOC( sizeof(int)*molp->natoms))) goto cleanup;
/* get just those H's which are capable of Hbonding */
if (! (hdonor = SYB_HBOND_DONORS( molp, NIL ) )) goto cleanup;

for (coord=ftemp, atyp=itemp, charge=ctemp, HAp=HAs, HDp=HDs, nat=1;
      nat<=molp->natoms; nat++)
{
    if (NIL == (at = SYB_ATOM_FIND_ID(molp, nat) ) ) goto cleanup;
    coord++ = at->xyz[0];
    coord++ = at->xyz[1];
    coord++ = at->xyz[2];
    *atyp++ = at->type - 1;
    *charge++ = at->charge;
    *HAp++ = SYB_ATAB_HBOND_ACCEPT(at->type) ;
    *HDp++ = UTL_SET_MEMBER(hdonor, at->recno) ;
}

for (b=0; b<regp->n_boxes; b++) {
    box = & regp->box_array[b];
    dohbd = (SYB_ATAB_ATOMIC_NUMBER( box->atom_type) == 1) &&
            (box->pt_charge == 1.0);
    dohba = (SYB_ATAB_ATOMIC_NUMBER( box->atom_type ) == 8) &&
            (box->pt_charge == -1.0);
    if (dohbd || dohba) {
        if (!TAILOR_STORE_IT_HERE( "TAILOR!FORCE_FIELD!HBOND_RAD_SCALING",
                                   &hbond_scal, 1)) goto cleanup;
        hbond_A = pow( hbond_scal, 6.0 );
        hbond_B = hbond_A * hbond_A;
    }
    if (vol_avg)
        QSAR_FIELD_EVAL_GETOFF(offs, box->stepsize, vol_avg, scale_vol_avg);
    if ( !no_st )
        QSAR_FIELD_VDWTAB ( box -> atom_type, repuls_val, ctp->du_lp_steric );
    for (iz=0, z=box->lo[2] ; iz < box->nstep[2]; iz++, z += box->stepsize[2])

```

```

for (iy=0, y=box->lo[1] ; iy < box->nstep[1]; iy++, y += box->stepsize[1])
  for (ix=0, x=box->lo[0] ; ix < box->nstep[0]; ix++, x += box->stepsize[0])
  {
    for ( coord = ftemp, charge = ctemp, atyp = itemp, HAp=HAs, HDp=HDs,
          do_steric=TRUE, do_elect=TRUE, nat=0, sum_steric = sum_elect = 0.0,
          nat<molp->natoms;
          nat++, wt++)
    {
      if ( ( *atyp == DUMMY-1 || *atyp == LP-1 ) && !ctp->du_lp_elect )
        *charge = 0.0; /* set charge to 0 since ignoring Du/Lp */
      if (!vol_avg) /* the "normal" case */
      {
        dis2 = x - *coord++;
        dis2 *= dis2;
        diff = y - *coord++;
        diff *= diff;
        dis2 += diff;
        diff = z - *coord++;
        diff *= diff;
        dis2 += diff;
        if ( !no_el && elfldp->zap_el==2 && do_elect) {
          dis = sqrt( dis2 );
          if ( dis < SYB_ATAB_VDW_RADII( *atyp+1 ) ) {
/* no shortcircuits! */
/*
 * elect++ = 0.0;
 * do_elect = FALSE;
*/
          }
        }
        if ( dis2 < MIN_SQ_DISTANCE ) {
          if ( !no_st )
            /* if atom has no steric value, we don't care about
               MIN_SQ_DISTANCE since it has no contribution anyway */
            if ( vdw_a[*atyp] != 0.0 && vdw_b[*atyp] != 0.0 ) {
              /* set sterics to its max value at current grid pt. */
              atm_steric = (*wt) * stfldp->max_value;
            }
          if ( !no_el && do_elect) {
            if ( !no_st && !do_steric && elfldp->zap_el ) {
              *elect++ = DAB_F_MISSING;
            }
            else if ( *charge != 0.0 ) {
              if ( *charge > 0.0 )
                atm_elect = (*wt) * elfldp->max_value;
              else atm_elect = (*wt) * -elfldp->max_value;
            }
          }
          if ( !do_elect && !do_steric )
            break; /* break out of loop since neither el. or st.
                     need to be calculated for this grid point */
/*
 * setting dis2 to 1 (an arbitrary no.) will prevent a zero
 * divide in the sum_steric or sum_elect calculations below */
        dis2 = 1.0;
      }
      if ( ! no_st && do_steric ) {
        dis6 = dis2 * dis2 * dis2;

```

```

dis12= dis6 ; dis6 ;
if (repulsive)
    dis12 = (repulsive==1) ? dis12 / dis2 : dis12 / dis2 / dis2;
if (dohbd && *HAp)
    atm_steric = hbond_B * vdw_b[*atyp]/dis12 -
        hbond_A * vdw_a[*atyp]/dis6 ;
else if (dohba && *HDp)
    atm_steric = hbond_B * vdw_b[*atyp]/dis12 -
        hbond_A * vdw_a[*atyp]/dis6 ;
else
    atm_steric = vdw_b[*atyp]/dis12 - vdw_a[*atyp]/dis6 ;
HAp++; HDp++; }
atm_steric = atm_steric > stfldp->max_value ? stfldp->max_value
    : atm_steric;
atm_steric *= (*wt);
if (!no_el && do_elect ) {
    atm_elect = *charge++ /
        ( dielectric ? sqrt(dis2) : dis2 ) ;
    atm_elect = atm_elect > elfldp->max_value ? elfldp->max_value
        : atm_elect;
    atm_elect = atm_elect < -(elfldp->max_value) ? -(elfldp->max_value)
        : atm_elect;
    atm_elect *= (*wt);
    sum_elect += atm_elect;
}
atyp++;
sum_steric += atm_steric;
}
else
{ for (off=0;off<9;off++)
}

coord += 3;
atyp ++ ;
charge ++ ;
HAp ++ ;
HDp ++ ;
}
} /* atom loop */
doneatoms:
if ( do_steric || do_elect ) {
    if (vol_avg) { sum_elect /= 9.0; sum_steric /= 9.0 ; }
    if ( !no_el && do_elect )
    { *elect = sum_elect * box->pt_charge * Q2KC ;
        if ( *elect > elfldp->max_value ) *elect = elfldp->max_value;
        else if ( *elect < - elfldp->max_value ) *elect =
            - elfldp->max_value;
        transform_field(elfldp->max_value,elect,ctp);
        elect++;
    }
    if ( !no_st && do_steric )
    { *steric = sum_steric ;
        if ( *steric > stfldp->max_value)
        { *steric = stfldp->max_value;
            if ( !no_el && elfldp->zap_el==1 ) *(elect-1) = DAB_F_MISSING; }
        transform_field(stfldp->max_value,steric,ctp);
        steric ++ ;
    }
}
} /* points in box loop */

```

```

} /* boxes loop */

    retval = TRUE;
cleanup:
    if ( itemp) UTL_MEM_FREE( itemp);
    if ( ftemp) UTL_MEM_FREE( ftemp);
    if ( ctemp) UTL_MEM_FREE( ctemp);
    if (HAs) UTL_MEM_FREE( HAs );
    if (HDs) UTL_MEM_FREE( HDs );
    if (hdonor) UTL_SET_DESTROY( hdonor );
    if (AtWts) UTL_MEM_FREE( AtWts );
    if (pset) UTL_MEM_FREE( pset );
    if (aset) UTL_MEM_FREE( aset );
return retval;

#undef Q2KC
#undef MIN_SQ_DISTANCE
}
/* -----
static fpt *QSAR_FIELD_RB_WTS( molp, rootid )
/* generates rotational-bond wts for each atom */
mol_ptr molp;
int rootid;
{
/* pseudo code for FIELD_RB_WTS()

    while saw new atoms
        uncover atoms that stopped last shell growth
        grow next "rotational shell"
        while adding to shell
            for each atom in shell
                get neighbors not seen
                for each neighbor
                    if bond is rotatable (acyclic, >1 attached atom, not =, am, #)
                        cover all other atoms attached to atom for this shell
                        add it to shell
*/
    fpt *ansr = NIL, *vals = NIL, factor, nowfact = 1.0;
    int found, aggcount, atid, aggid, loop, size;
    set_ptr aggats = NIL, allats = NIL, nuls = NIL, endatms = NIL, end_cands
    atom_ptr root, SYB_ATOM_FIND_REC(), at, atrec ;
    bond_ptr b, SYB_BOND_FIND_REC();
    List_Ptr toats, UTL_LIST_RETRIEVE_P();
    acon_ptr cptr;
    char tempString[200];
    void ashow(), qsar_field_attached_atoms();

    if (! ( vals = (fpt *) UTL_MEM_ALLOC( sizeof(fpt)*molp->natoms))) return( NI
    if (!UIMS2_VAR_GET_TOKEN( "TAILOR!COMFA!AGGREG_DESCALE",
        &factor ) ) return( NIL );
    if (! (aggats = UTL_SET_CREATE( molp->max_atoms + 1 ) )) goto cleanup;
    if (! (allats = UTL_SET_CREATE( molp->max_atoms + 1 ) )) goto cleanup;
    if (! (nuls = UTL_SET_CREATE( molp->max_atoms + 1 ) )) goto cleanup;
    if (! (endatms = UTL_SET_CREATE( molp->max_atoms + 1 ) )) goto cleanup;
    if (! (end_cands = UTL_SET_CREATE( molp->max_atoms + 1 ) )) goto cleanup;
    if (! ( root = SYB_ATOM_FIND_REC( molp, rootid ) )) goto cleanup;
    UTL_SET_INSERT( aggats, root->recno );
    UTL_SET_INSERT( allats, root-> recno );
    aggcount = loop = 1;

```

```

while (TRUE) {
    while (TRUE) {
        aggid = -1;
        while ((aggid = UTL_SET_NEXT( allats, aggid )) >= 0 ) {
            UTL_SET_CLEAR( nuls );
            qsar_field_attached_atoms( nuls, molp, aggid );
            UTL_SET_DIFF_INPLACE( nuls, allats, nuls );
            UTL_SET_DIFF_INPLACE( nuls, endatms, nuls );
/* identifying any atoms that terminate this aggregate */
        atid = -1;
        while ((atid = UTL_SET_NEXT( nuls, atid )) >= 0 ) {
            if (!!( at = SYB_ATOM_FIND_REC( molp, atid ) )) goto cleanup;
/* skipping monovalent atoms */
            if (at->nbond > 1) {
/* find bond record that attaches to aggid */
                toats = at->conn_atom;
                found = FALSE;
                while (toats && !found ) {
                    toats = UTL_LIST_RETRIEVE_P( toats, &cptr, &size );
                    found = (cptr->target == aggid );
                }
                if (!found) goto cleanup;
                b = SYB_BOND_FIND_REC (molp, cptr->bond_rec);
                if ( !(b->status & BOND_V_IRING) && !(b->status & BOND_V_ERI)
                    && (b->type == SYB_BTAB_MNEM_TO_TYPE("1") ) ) {
/* have an end-of-aggregate atom, mark as end atoms all other attached atoms */
                    UTL_SET_CLEAR( end_cands );
                    qsar_field_attached_atoms( end_cands, molp, at->recno );
                    UTL_SET_DELETE( end_cands, aggid );
                    UTL_SET_OR_INPLACE( endatms, end_cands, endatms );
                }
            }
            UTL_SET_OR_INPLACE( aggats, nuls, aggats );
        }
        if (UTL_SET_CARDINALITY( aggats ) <= aggcount ) break;
        aggcount = UTL_SET_CARDINALITY( aggats );
        UTL_SET_OR_INPLACE( allats, aggats, allats );
    }
/* debugging stuff .. */
/*
    sprintf( tempString, "Aggregate %d (weight = %f ):", loop, nowfact );
    UBS_OUTPUT_MESSAGE( stdout, tempString );
    ashow( aggats, molp );
*/
/* if no atoms added, we are done! */
    if (UTL_SET_EMPTY( aggats )) break;
/* record scaling factor for atoms in this aggregate */
    atid = -1;
    while ((atid = UTL_SET_NEXT( aggats, atid )) >= 0 ) {
        if (!(atrec = SYB_ATOM_FIND_REC( molp, atid ))) goto cleanup;
        vals[ (atrec->id)-1 ] = nowfact;
    }
    UTL_SET_OR_INPLACE( allats, aggats, allats );
    UTL_SET_CLEAR( aggats );
    UTL_SET_CLEAR( endatms );
    aggcount = 0;
    nowfact *= factor;
    loop++;
}

```

```

ansr = vals;

cleanup:
    if (aggats) UTL_SET_DESTROY( aggats );
    if (allats) UTL_SET_DESTROY( allats );
    if (endatms) UTL_SET_DESTROY( endatms );
    if (end_cands) UTL_SET_DESTROY( end_cands );
    if (nuls) UTL_SET_DESTROY( nuls );
    return( ansr );
}

static void qsar_field_attached_atoms( aset, m, atid )
/* ors atoms attached to atm into aset */
/* WORKS STRUCTLY WITH RECNOS */
set_ptr aset;
mol_ptr m;
int atid;
{
    atom_ptr at, SYB_ATOM_FIND_ID();
    List_Ptr tohs, UTL_LIST_RETRIEVE_P();
    atom_ptr toh, SYB_ATOM_FIND_REC();
    acon_ptr conn1;
    int nbytes1;

    at = SYB_ATOM_FIND_REC( m, atid );
    tohs = at->conn_atom;
    while (tohs) {
        tohs = UTL_LIST_RETRIEVE_P( tohs, &conn1, &nbytes1 );
        toh = SYB_ATOM_FIND_REC( m, conn1->target );
        UTL_SET_INSERT( aset, toh->recno );
    }
    return;
}

static void ashaw( aset, m )
/* for interactive debugging, shows a set's membership in terms of atom ID */
set_ptr aset;
mol_ptr m;
{
    char buff[1000], *b;
    atom_ptr at, SYB_ATOM_FIND_REC();
    int elem;

    *buff = '/0';
    b = buff;
    elem = -1;
    while ( (elem = UTL_SET_NEXT( aset, elem)) >= 0 ) {
        at = SYB_ATOM_FIND_REC( m, elem );
        sprintf( b, "%d", at->id );
        b = buff + strlen( buff );
    }
    sprintf( b, "\n" );
    UBS_OUTPUT_MESSAGE( stdout, buff );
}

```

```

#
# Section II-A. SPL invoked shell for computing the diagonal defining the
# "best" triangle, e.g., the one with the highest density of points below.
#
# @expression_generator LRT_FAST
# Usage:
#   lrt_fast rows descriptor_cols bio_col [pls flags like scaling in quotes]
#       rows (*) - rows to take
#       descriptor_cols - which columns are the neighborhood metrics
#       bio_col - which column has the bio (probably log bio) data
#       [...] - if need to SCAL NONE or anything like that, do it here
#
# returns a line of the form
#   3.09691 / 0.000546509 = 5666.71 - 496 : 496 :: 15.6981 : 15.6989
#       ^ max bio difference
#           ^ optimal distance division for max bio
#               ^ slope
#                   ^ number in the lrt
#                   ^ total number
#                       ^ area in the lrt
#                           ^ total area
#
# Significance is related to whether ratio of numbers is
# much above ratio of areas.
#
globalvar SAMPLS_IN_PROGRESS DONE_CHECKED_OUT
localvar hold distname rows cols bio
setvar rows %promptif("$1" ROW_EXP "*" "Rows to use in lrt")
setvar cols %promptif("$2" COL_EXP "COMFA*" "Columns of mol descriptors")
setvar bio %promptif("$3" COL_EXP "LOGBIO" "Column of bio data")
setvar hold SAMPLS_IN_PROGRESS
setvar SAMPLS_IN_PROGRESS $bio
setvar distname TAILOR!HIER!DIST_FNAME
setvar TAILOR!HIER!DIST_FNAME lrt_fort.3
#
# here the information is computed and written to a file
# whose name is passed in via a TAILOR value
QSAR ANA DO I >$NULLDEV $rows $cols HIER $4 |
setvar SAMPLS_IN_PROGRESS $hold
setvar TAILOR!HIER!DIST_FNAME $distname
#
# contents of the file are returned to the caller
setvar hold %system("cat lrt_fort.3")
%return( "$hold" )
#
# Section II-B. SPL script for computing the significance of the distribution
# found by lrt_fast
#
@expression_generator dochi
# computes the chi-square statistic for the number of points below
# the diagonal, null hypotheses being the area fraction of the total.
#
# To be called as: %dichi( %lrt_fast( ) ), i.e., its inputs

```

```

# are exactly the input of %lrt_fast as described in the lrt_fast header.
#
setvar expected %math( $9 * $11 / $13 )
setvar sq %math( $7 - $expected )
setvar sq %math( $sq * $sq / $expected )
%return( $sq )

/*
Section II-C. Computes the best diagonal in the "virtual graph" of biological
distances vs property differences. */

int QSHELL_HIER_LRT(table,biocol,dmat,nrow,order,lmsg)
char *table;
int biocol, /* column in MSS with biological data */
nrow, /* dimension of dmat and order */
*order; /* array of row IDs to consider */
fpt *dmat; /* distance matrix for property distances */
char *lmsg; /* file name for results */
{
fpt *p, *q, fabs(), bmax;
int i,j, count, status_array;
char *fpt_colname;
FILE *out, *UTL_FILE_FOPEN();

/* need to get the bio values
In the n^2 we can repack into n(n-1)/2 then add the n bio values
and finish with the bio distances */
/* */
/* No error handling. Better be data in those rows!
*/
for(count=0, i=0; i<nrow; i++)
for (j=0; j<i; j++)
dmat[count++] = dmat[i*nrow + j];
q = p = dmat + ( (nrow-1) * nrow ) / 2;
TBL_ACCESS_INDEX_TO_COLNAME(table, biocol-1, &fpt_colname);
TBL_GRAB_INIT_FPTS(table, 1, &fpt_colname );
for ( i=0;i<nrow;i++, p++)
TBL_GRAB_GET_FPTS_INV(order[i]-1, &status_array, p);
TBL_GRAB_COMPLETE_FPTS();

bmax = 0.0;
for (count=0, i=0; i<nrow; i++)
for (j=0; j<i; j++, count++)
if ( (p[count] = fabs(q[i] - q[j])) > bmax) bmax = p[count];

out = UTL_FILE_FOPEN(lmsg, "w");
QSHELL_HIER_DO_LRT(out,count,dmat,p, bmax);

UTL_FILE_FCLOSE(out);
}

```

```

int QSHELL_HIER_DORT( out, index, xsort, ysort, bmax )
FILE *out;
fpt *xsort, *ysort, bmax;
int index;
{
    int *order, count, j, i, bad;
    int bestN, bestI;
    fpt den,bestDen;

#define CUTOFF ( bmax * ( xsort[order[i]] / xsort[order[j]] ) )

    if (! (order = (int *) UTL_MEM_ALLOC( index *sizeof(int) ))) return 0;
    for (i=0;i<index;i++) order[i]=i;
    bestN = bestI = bad = 0;
    bestDen = 0.0;

    fpt_heapsort(index, xsort, order);

    for (j=0;count=0, bad=0, j<index ;j++)
    {
        if (xsort[order[j]] <= 0.0) continue;
        for (i=0;i<=j;i++)
        {
            if (ysort[order[i]] <= CUTOFF) count++;
            else                                bad++;
            /* loop over all d <= this distance */
            if ( (den = count/ bmax / xsort[order[j]] *2.0) > bestDen)
                {bestDen = den; bestI = j; bestN = index - bad;}
        } /* loop over all distances */
        den = bmax * xsort[order[index-1]];
        sprintf(msg,"%g / %g = %g - %d : %d :: %g : %g\n",
               bmax,xsort[order[bestI]], bmax/xsort[order[bestI]],
               bestN,index,den-xsort[order[bestI]]*bmax/2.0, den);
        UBS_OUTPUT_MESSAGE(out,msg);
        UTL_MEM_FREE(order);
        return 1;
    }
}

```

```

/* n is number of elements
   arrin is array of floats to be sorted
   indx is array of ints initially 0...n-1
*/
int fpt_heapsort(n,arrin,indx)
int n;
fpt *arrin;
int *indx;
{
    int l, ir, indxxt, i, j;
    fpt q;

    l = n/2 ;
    ir = n -1 ;

    while (TRUE)      /* the "10" loop */
    {
        if (l>0) { indxxt = indx[--l]; q = arrin[indxxt]; }
        else
        {
            indxxt = indx[ir]; q = arrin[indxxt];
            indx[ir--] = indx[0];
            if ( ir == 0 )
                { indx[0] = indxxt; return 1; } /* <== Only way out ! */
        }

        i = l;
        i = l;
        j = l + 1 + 1;
        while (j <= ir) /* the "20" loop */
        {
            if ( (j<ir) && (arrin[indx[j]] < arrin[indx[j+1]]) ) j++ ;
            if (q < arrin[indx[j]]) { indx[i] = indx[j]; i = j; j = j+j+1; }
            else
                { j = ir+1; }
        }

        indx[i] = indxxt;
    }
}

```

/\* SECTION III-A. Declarations for all non-standard data structures referenced  
in the C code functions shown in Sections I and II. \*/

```

/********************* Molecule and Supporting Structure Definitions *****/
/* John McAlister 09-Aug-1985 */
/*
 * This file contains the definitions for the molecular data structures required within SYBYL. The contents of this file are described in detail in the document "SYBYL Molecular Data Structures".
 */
/********************* Define the molecule descriptor template *****/
/* typedef struct molecule_struct {
    char      *name;          /* pointer to molecule name */
    i32       type;           /* molecule type */
    List_Ptr  dict;           /* list of dictionaries used with molecule */
    i32       status;          /* molecule status */
    char      *comment;        /* pointer to comment for molecule */
    stamp     cre_time;        /* creation time/user/version stamp */
    stamp     mod_time;        /* modification time/user/version stamp */
    int       max_props;       /* maximum properties currently allocated */
    int       nprops;          /* number of molecular properties */
    List_Ptr  props;           /* pointer to list of properties */
    int       max_feats;        /* maximum features currently allocated */
    int       nfeats;          /* number of molecular features */
    List_Ptr  feats;           /* pointer to list of molecular features */
    int       max_subst;        /* maximum substructures currently allocated */
    int       nsubst;          /* number of substructures in molecule */
    List_Ptr  subst;           /* pointer to list of substructures */
    List_Ptr  subst_roots;     /* pointer to list of root subst offsets */
    int       max_atoms;        /* maximum atoms currently allocated */
    int       natoms;          /* number of atoms in molecule */
    List_Ptr  atoms;           /* pointer to atom array segment list */
    int       max_bonds;        /* maximum bonds currently allocated */
    int       nbonds;          /* number of bonds in molecule */
    List_Ptr  bonds;           /* pointer to bond array segment list */
    int       charges;          /* type of atomic charges, if present */
    fpt      vector[3];         /* translation vector for molecule */
    fpt      matrix[9];         /* rotation matrix for molecule */
    List_Ptr  assoc_data;       /* pointer to list of associated data */
                                /* descriptors */
} molecule, *mol_ptr;
*/
/********************* ATOM DEFINITION *****/
/* Define the atom entry record template */
typedef struct atom_struct {
    char      *name;           /* atom name */
    int       type;            /* atom type */
    i32       status;          /* atom status */
    int       recno;           /* cumulative atom record number */
    int       id;              /* atom id (logical atom number) */
    int       link;             /* link to next atom record */
    int       subst;            /* offset to substructure containing atom */
    List_Ptr  property;        /* pointer to list of properties for atom */
    List_Ptr  feature;         /* pointer to list of features including
                                /* this atom */
    int       nbond;           /* number of bonds involving this atom */
}

```

```

List_Ptr conn_atom; /* pointer to list of bonded atoms */
fpt xyz[3]; /* coordinates of atom */
fpt charge; /* point charge on atom */
} atom, *atom_ptr;

/* Define the atom array segment descriptor template */
typedef struct atom_seg_struct {
    atom_ptr seg_head; /* pointer to head of atom array segment */
    mol_ptr molecule; /* pointer to molecule containing atom seg */
    int max_atom; /* maximum number of atom records in seg */
    int natom; /* number of filled atom records in seg */
    int used_atom; /* offset to first filled record in segment */
    int free_atom; /* offset to first free record in segment */
} atom_seg, *aseg_ptr;

/* Define the bond specifier records pointed to by the atom records */
typedef struct atom_conn_struct {
    int target; /* offset to target atom */
    int bond_rec; /* offset to bond descriptor record */
} atom_conn, *acon_ptr;

```

```

***** BOND DEFINITION *****
/*
/* Define the bond entry record template
typedef struct bond_struct {
    int      type;          /* bond type */          */
    i32     status;         /* bond status */        */
    int      recno;         /* cumulative bond record number */
    int      id;            /* bond id (logical bond number) */
    int      link;          /* link to empty bond record */
    List_Ptr property;     /* pointer to bond property list */
    List_Ptr feature;      /* pointer to list of features including
                           /*   this bond */
    int      o_subst;       /* offset to origin atom substructure */
    int      origin;         /* offset to atom at bond origin */
    int      t_subst;       /* offset to target atom substructure */
    int      target;         /* offset to atom at bond destination */
} bond, *bond_ptr;
*/
/* Define the bond array segment descriptor template
typedef struct bond_seg_struct {
    bond_ptr seg_head;     /* pointer to head of bond array segment */
    mol_ptr  molecule;     /* pointer to molecule containing bond seg */
    int      max_bond;     /* maximum number of bonds in segment */
    int      nbond;         /* number of filled bond records in seg */
    int      used_bond;     /* offset to first filled record in segment */
    int      free_bond;     /* offset to first free record in segment */
} bond_seg, *bseg_ptr;
*/

```

```

/********************* comfa.h *****/
/* Regions are the set of points at which energy evaluations are made */
/* in the CoMFA method of QSAR. A region is defined as the union */
/* of a set of 3D boxes (which may be a single point in the */
/* limit) and their associated attributes. Attributes needed for */
/* CoMFA purposes are outlined below. */
/********************* */

#ifndef QSAR_COMFA_DEFINITIONS
#define QSAR_COMFA_DEFINITIONS 1
#include "ta_types.h"
#define DUMMY 26      /* dummy atom id */
#define LP    20      /* lone pair atom id */

typedef enum {
    FDENGY_UNKNOWN,
    FDENGY_ELECT,
    FDENGY_STERIC,
    FDENGY_HOMO,
    FDENGY_LUMO,
    DOCK_ELECT,
    DOCK_STA_NOHB,
    DOCK_STA_HBD,
    DOCK_STA_HBA,
    DOCK_STB_NOHB,
    DOCK_STB_HBD,
    DOCK_STB_HBA } FldEngyTyp;

typedef enum {
    FDHD_ORIGINAL,
    FDHD_FFIT,
    FDHD_XTERN,
    FDHD_FUNC,
    FDHD_USER,
    FDHD_USR_AVG,
    FDHD_DOCK,
    FDHD_AVG,
    FDHD_SIG,
    FDHD_MAX,
    FDHD_MIN,
    FDHD_COEFF,
    FDHD_AVG_X,
    FDHD_SIG_X,
    FDHD_FLD_X,
    FDHD_RANGE,
    FDHD_PLS_XWT,
    FDHD_PLS_XLOAD,
    FDHD_FAC_LOAD,
    FDHD_FAC_COMM,
    FDHD_FAC_ROTLOAD,
    FDHD_SIMCA_LOAD,
    FDHD_SIMCA_MODEL,
    FDHD_SIMCA_DISCRIM,
    FDHD_HBD } FldHowTyp;

```

```

typedef struct {
    fpt lo[3],           /* corner with lowest values for each axis      */
    hi[3],           /* "      "      "      hi-est      "      "      "      */
    stepsize[3];       /* increment between points                      */
    int nstep[3],       /* derived as 1 + (hi-lo + epsilon) / stepsize */
    n;                /* n = product of nstep[i]                      */
    int atom_type;     /* SYBYL atom type, for steric energy computation */
    fpt pt_charge;    /* elemental charge at point, for electrostatics   */
    fpt *weight;       /* weight[n] is applied in all computations, e.g.=1 */
    int avg_type;     /* box of 'scale', sphere, sphere x vdw, ...?      */
    fpt avg_scale;    /* scale whose meaning derived from avg_type      */
    int arb;           /* arbitrary int for later use                   */
    *parb;            /*          "      pointer      "      "      */
} Box, *BoxPtr ;
}

typedef struct {
    char *filename ;    /* name of the region's file (if any)           */
    int n_boxes;        /* number of boxes which make up the region      */
    int n_points ;     /* number of points in this region altogether    */
    BoxPtr box_array;  /* box_array[n_regions], each one a Box          */
    int n_refs ;        /* number of CURRENT references to this memory   */
    long when_made;    /* creation stamp                                */
} Region, *RegionPtr ;
}

typedef struct {
    char *reg_name;    /* name of the region's file (if any)           */
    char *fld_name;    /* name of this field's file (if any)           */
    RegionPtr reference; /* the region referenced by this field          */
    FldEngyTyp fld;   /* what type of field is referenced here        */
    int num_avgd;      /* number of fields averaged into this one       */
    int curr_iter;     /* number of iterations in current field fit run */
    char *mol_id;      /* unspecified molecule id,
                           e.g. dbname/molname/alignname               */
    int n_points ;     /* number of points in associated region         */
    int zap_el;         /* whether electrostatics are MISSING when>max_st */
    fpt max_value;     /* largest permitted absolute value of energy    */
    fpt *field_value;  /* values at each point of the field             */
    int n_refs ;        /* number of CURRENT references to this memory   */
    long when_made;    /* creation stamp                                */
    int vol_avg_type;  /* added these 4 items 1/30/89 DEP */
    fpt scale_vol_avg;
    int dielectric;
    int repulsive;
    FldHowTyp how_made; /* perry's way = 1 or old way = 0 */
} Field, *FieldPtr ;
}

```

```

/* molecule dependent information solicited by QSAR table operations,
   passed into COMFA column field evaluations */
```

```

typedef struct {
    boolean already_field; /* whether a field name exists (otherwise alignment) */
    char *some_name; /* name of alignment; NIL align==use as is (!) */
    char *steric_name; /* name of steric field (if applicable) */
    char *elect_name; /* name of electrostatic field (if applicable) */
    FieldPtr sfld_p; /* points to steric field in memory (when there) */
    FieldPtr efld_p; /* points to elect. field in memory (when there) */
} ComfaMol, *ComfaMolPtr;
```

```

/* molecule-independent information for CoMFA evaluations */

typedef struct {
    int vol_avg ; /* case for volume averaging: 0,1,2=none,box,sphere(0) */
    fpt vol_scale ; /* scale for volume averaging (1.0) */
    int fld_types ; /* case for what fields: 0,1,2=both,steric,elect.(0) */

    fpt steric_max; /* maximum steric energy (30) */
    int repulsive ; /* steric repulsive exponent - 12,10,or 8 (12) */
    fpt elect_max ; /* maximum electrostatic energy (30) */
    int dielectric; /* case for dielectric (AS FORCE FIELD TAILOR) */
    int elect_out ; /* case to drop elect inside steric max: 0,1=T,F (1) */

    char *region_name; /* name of region used in the CoMFA computations */

    FieldPtr sweight_fld; /* points to MEMORY field for weighting steric PLS */
    FieldPtr eweight_fld; /* points to MEMORY field for weighting elect. PLS */
    FldHowTyp how_done; /* perry's way = 1 or old way = 0 */
    int du_lp_steric; /* include dummies and lone pairs in steric field
                       calculations */
    int du_lp_elect; /* include dummies and lone pairs in electrostatic
                      field calculations */
    int spare1; /* As of 6.1comfa , this is TAILOR!COMFA!TRANSFORM*/
    int spare2; /* INDICATOR SCALE among other things */
} ComfaTop, *ComfaTopPtr;
```

#endif

Section III-B. Functional descriptions of external procedures.  
(Routines that simply return dynamic memory to the heap are not described.)

BOND\_V\_ERING - TRUE if bond is in an external ring.

BOND\_V\_IRING - TRUE if bond is in an internal (simple) ring.

QSAR\_FIELD\_EVAL\_GETOFF - provides coordinates for field computation when "volume averaging" is being done.

QSAR\_FIELD\_VDWTAB - returns steric parameters for the computation of the field contribution from the probe atom and each of the molecule atoms.

SYB\_AREA\_GET\_MOLECULE - returns the internal representation of the molecule in some area or "container", if such exists.

SYB\_ATAB\_ATOMIC\_NUMBER - returns the atomic number of the specified atom type.

SYB\_ATAB\_ATOMIC\_WEIGHT - returns the atomic weight of the specified atom type.

SYB\_ATAB\_HBOND\_ACCEPT - returns TRUE if the specified atomic type is a hydrogen-bond accepting atom.

SYB\_ATAB\_VDW\_RADII - returns the atomic radius of the specified atomic type.

SYB\_ATOM\_FIND\_ID - returns the internal representation of an atom referenced by its atom ID number (Atom IDs are guaranteed to be continuous but the ID of any single atom may change as atoms are added or deleted.)

SYB\_ATOM\_FIND\_REC - returns the internal representation of an atom referenced by its record ID number. (Atom record IDs are invariant but there may be "holes" in their sequence such that the largest record ID may be greater than the number of atoms.)

SYB\_ATOM\_FIND\_SET - returns the bitset of atoms corresponding to a list of atoms.

SYB\_BOND\_FIND\_REC - returns the internal representation of a bond referenced by its (invariant) record ID number.

SYB\_BTAB\_MNEM\_TO\_TYPE - converts an ASCII representation of a bond type to its internal representation.

SYB\_EXPR\_ANALYZE - parses a user-entered ASCII description of atoms (e.g., M2(<H>)) for all hydrogen atoms within molecule M2) into internally valid representations of molecule and atoms.

SYB\_HBOND\_DONORS - returns the set of IDs for atoms which are hydrogen-bonding hydrogens.

TAILOR\_STORE\_IT\_HERE - returns the current value of a user- (and SPL-) accessible variable.

TBL\_ACCESS\_INDEX\_TO\_COLNAME - converts a user-provided MSS column ID to a column name (name is guaranteed to be a unique identifier).

TBL\_GRAB\_COMPLETE\_FPTS - done returning multiple (scalar) values in an MSS column to an array.

TBL\_GRAB\_GET\_FPTS\_INV - in a multiple value retrieval, returns the value corresponding to a user-provided row ID.

TBL\_GRAB\_INIT\_FPTS - set up for returning multiple (scalar) values in an MSS column to an array.

UBS\_OUTPUT\_MESSAGE - equivalent to *fprintf()*

UIMS2\_VAR\_GET\_TOKEN - returns the current value of a global SPL variable.

UIMS2\_WRITE\_ERROR - writes text to the error output stream.

UTL\_FILE\_FCLOSE, UTL\_FILE\_FOPEN - equivalent to *fclose()* and *open()*.

UTL\_LIST\_RETRIEVE - returns the next element on a linked list.

UTL\_MEM\_ALLOC - equivalent to *malloc()*.

UTL\_SET\_AND\_INPLACE -- makes the first set logically equivalent to the second set, with only those bits that are also 1 in the third set becoming 1 in the first set.

UTL\_SET\_CARDINALITY - returns the number of bits that are 1 in a particular bitset.

UTL\_SET\_CLEAR - sets all bits in the set to 0.

UTL\_SET\_COPY\_INPLACE - makes the first set logically identical to the second.

UTL\_SET\_CREATE - creates and returns an empty set of requested size.

UTL\_SET\_DELETE - sets the specified bit to 0.

UTL\_SET\_DIFF\_INPLACE - makes the first set logically equivalent to the second set, with all bits that are 1 in the third set becoming 0 in the first set.

UTL\_SET\_EMPTY - TRUE if all bits in the set are 0.

UTL\_SET\_INSERT - sets the requested bit to 1.

UTL\_SET\_MEMBER - returns TRUE if the requested set bit equals 1.

UTL\_SET\_NEXT - returns the identity of the next non-zero bit in a set.

UTL\_SET\_OR\_INPLACE - makes the first set logically equivalent to the second set, with all bits that are 1 in the third set becoming 1 in the first set.

UTL\_STR\_CMP\_NOCASE - non-case sensitive version of strcmp().

## APPENDIX "B"

```
/* CODE. This code implements a PHORE_LOC column type and calculates a single
cell value (the Hydrogen Bonding Fingerprint for a molecule) within the SYBYL
Molecular Spreadsheet. It is to be understood that other supporting code handles
user input, user output, and disk file I/O. */

/* data structure for PHORE_LOC column type */
typedef
    struct PHORE {
        char *disco_fn; /* user name for DISCO feature file - default
appears below */
        int disco_in; /* internal flag if DISCO feature file loaded */
        char *region_fn; /* user name for defining region file */
        RegionPtr rgn; /* internal reference to region when loaded */
        int nfuzz; /* number of extra lattice points (each direction)
for each PHORE feature */
        int nbits; /* set length (must agree with rgn contents or EVAL
fails) */
    } PHORE, *PPHORE;

/*+E* QSAR_PROC_EVAL_PHORE_LOC */
***** */
/*     int QSAR_PROC_EVAL_PHORE_LOC(tablename, row, colname) */
/* */
/*      Dick Cramer 31-Jul-95      (PHORE_LOC == lattice bitset ) */
/* */
/* This module generates bitsets whose cardinality is equal to
lattice points x 2 (# of sitepoint classes. For each
instance of a pharmacophoric point in the molecule being
processed, the geometrically nearest (1+m)^3 bits in the
bitset will be set to 1 (where m is user supplied).
*/
/* NOTE: this routine explicitly requires that sets begin after a
first element that is the set size!!!
*/
/* Inputs
*/
/* Outputs
*/
/* User Required Definition Files
*/
***** */
/*-E*/
int QSAR_PROC_EVAL_PHORE_LOC(tablename, row, colname)
char *tablename, *colname;
int row;
```

```

{
    mol_ptr      mol;
    PPHORE      phr;
    int         err, status, nvalid, mol_area;
    char        *dum;
    set_ptr     print, qsar_proc_calc_phore_set();
    FILE *fp;

/* get the molecule */
    if ( !TBL_UTL_GET molecule(tablename, row, FALSE, &mol) )
    {
        if ( UTL_ERROR_IS_SET() )                                {err=1; goto
error;}
        else return FALSE;
    }

/* get the user-provided input data */
    if ( !TBL_ATTR_FIND_COLUMN_A(tablename, colname, "PROC_SUPPORT", &dum,
                                  (int *)&phr) )           {err=3; goto
error;}
/* retrieve DISCO stuff if not yet present */
    if ( !phr->disco_in) {
        if ( !phr->disco_fn) {err=1; goto error;}
/* set appropriate tailor value, then initialize DISCO */
        sprintf( str, "SETVAR TAILOR!DISCO!FILE %s", phr->disco_fn );
        QUILMS2_EXEC_COMMAND( str );
        QUILMS2_EXEC_COMMAND( "DISCO INIT" );
        phr->disco_in = TRUE;
    }

/* retrieve region if not yet present */
    if (!phr->rgn) {
        if ( !phr->region_fn) {err=1; goto error;}
        if ( !(phr->rgn = QSAR_REGION_RETRIEVE( phr->region_fn ) ) )
{err=4;goto error;}
        if (phr->rgn->n_boxes > 1) {
            sprintf( str, "WARNING: Region %s has %d boxes. Only first
will be used.\n",
                    phr->region_fn, phr->rgn->n_boxes );
            UBS_OUTPUT_MESSAGE( stdout, str );
        }
        phr->nbits = 2 * phr->rgn->n_points;
    }

/* evaluate this result, first the DISCO call */
    if ( !( print = qsar_proc_calc_phore_set( mol, phr, &nvalid ) ) ) {err=12;
goto error;}

/* go store both the bitset in the MSS "Cell_Support" and the number of bits
actually set in the "CELL", so there's something for the user to see */
    if ( !TBL_ACCESS_X_PUT_VALUE(tablename, row, colname, "CELL_SUPPORT",
                                 (int *)&print) )           {err=11; goto error;}
}

```

```

if ( !TBL_ACCESS_X_PUT_VALUE(tablename, row, colname, "CELL",
                               (int *)&nvalid) )           {err=11; goto
error;}
    return TRUE;

error:
    sprintf (str, "QSAR_PROC_EVAL_PHORE_LOC (%d)", err);
    UTL_ERROR_ADD_TRACE(str);
    return FALSE;
}

set_ptr qsar_proc_calc_phore_set( mol, phr, nvalid )
/* creates actual bitset */
{
    mol_ptr      mol;
    PPHORE       phr;
    int          *nvalid;
{
    set_ptr anset = NIL, pset = NIL, SYB_FEAT_FIND_ID_SET();
    feat_ptr featp, SYB_FEAT_FIND_REC();
    atom_ptr   a, SYB_ATOM_FIND_REC();
    int     err, elem, sitebase, ci, xybase, boff, lt_base[3], lt_off[3], loff =
0, hioff = 0 ;
    fpt      tmp;
    BoxPtr      bpxptr;
    Line_ptr    cdp;
}
    if (! ( anset = UTL_SET_CREATE( phr->nbits ) ) ) {err = 1; goto error;}
    *nvalid = 0;
    if (phr->nfuzz) {
        loff -= phr->nfuzz / 2;
        hioff += (phr->nfuzz + 1 ) / 2;
    }
    bpxptr = phr->rgn->box_array;
    xybase = bpxptr->nstep[0] * bpxptr->nstep[1];
/* generate the DISCO sites for this molecule, which .. */
    UIMS2_EXEC_COMMAND( "ECHO %DISCO_SITES()" );
/* .. become "FEATURES" + "dummy atoms" within SYBYL's molecule data
structure */
    pset = SYB_FEAT_FIND_ID_SET(mol, FEAT_V_LINE, 1, mol->nfeats);

    if (pset) {

        elem = -1;
        while((elem = UTL_SET_NEXT(pset,elem)) != NO_MORE_ELEM) {
            if (!(featp = SYB_FEAT_FIND_REC (mol,elem))) goto error;
            if ((featp->name[1] == 'S') && (featp->name[2] == '_')) {
/* have an H-bonding feature, it must represent a line */
                sitebase = featp->name[0] == 'A' ? 0 : phr->rgn->n_points;
/* the dummy atom at the end of the line is our H-bonding locus */

```

```

    cdp = (line_ptr) featp->dataptr;
    if (!(a = SYB_ATOM_FIND_REC (mol, cdp->positn)) ) {err=2; goto
error;}
    for (ci = 0; ci < 3; ci++ ) {
        tmp = (a->xyz[ci] - bptr->lo[ci]) / bptr->stepsize[ci];
        lt_base[ci] = (int) (tmp < 0.0 ? tmp - bptr->stepsize[ci] :
tmp );
    }
/* cycle through all points touched by this locus that are also within the
region */
    for (lt_off[0] = lt_base[0] + loff; lt_off[0] <= lt_base[0] + hioff;
lt_off[0]++)
        if (lt_off[0] >= 0 && lt_off[0] < bptr->nstep[0])
            for (lt_off[1] = lt_base[1] + loff; lt_off[1] <= lt_base[1] +
hioff; lt_off[1]++)
                if (lt_off[1] >= 0 && lt_off[1] < bptr->nstep[1])
                    for (lt_off[2] = lt_base[2] + loff; lt_off[2] <= lt_base[2] +
hioff; lt_off[2]++)
                        if (lt_off[2] >= 0 && lt_off[2] < bptr->nstep[2] ) {
                            boff = xybase * lt_off[2] +
                                (bptr -> nstep[0]) * lt_off[1] +
                                lt_off[0] + sitebase;
                            UTL_SET_INSERT( anset, boff );
                            (*nvalid)++;
                        }
                }
}
UTL_SET_DESTROY( pset );
/* pset exists */

return( anset );

error:
sprintf (str, "qsar_proc_calc_phore_set(%d)", err);
UTL_ERROR_ADD_TRACE(str);
return FALSE;
}

```

```

# This file determines the recognition of site points in Sybyl/DISCO.
# See the SYBYL DISCO manual for detailed documentation. The defined types
are
# (1) HB : the QUERY is searched in the SEARCH mode, and all occurrences
# are assigned DISCO features according to the remaining
# specifications -- the three ATOMS refer to the atom number
# in QUERY such that the feature is DIST from the first atom
# at bond ANGLE with the first and second atom at each of the
# TORSIONS formed by the site point and the three ATOMS in order.
# A sitepoint of NAME is added at these extension points,
# -- and -- the first atom is assigned a feature complimentary

```

# to the extension point (such as HBD\_CO\_ and RHBD\_CO\_).  
# (2) HBex:differs from HB in that the angles and torsions are replaced  
# by two other arguments: whether lone pairs are part of the  
# extension point placement, and which ATYPE (generally LP  
# and/or H) determine the direction of the sitepoints.

#TYPE	NAME	ATOMS	SEARCH	DIST	ANGLE	TORSIONS	QUERY
=====	=====	=====	=====	=====	=====	=====	=====
HB	DS_O2C2_	4 2 1	NoDup	2.9	120	"0.0 180.0"	HevC(Any)=O[f]
HB	DS_O3Car_	1 3 4	All	2.9	119	"0.0 180.0"	O[f]HC(:Hev):Hev
HB	DS_O3Car_	1 2 3	All	2.9	119	"0.0 180.0"	O[f]C(:Hev):Hev
HB	DS_O3Car_	1 3 4	NoDup	2.9	119	"0.0 180.0"	O[f]HC(=O)
HB	DS_O3Car_	1 2 3	NoDup	2.9	119	"0.0 180.0"	O[f]C(=O)
HB	DS_O3Car_	2 1 3	All	2.9	120	"0.0 180.0"	C(:O[f]):O[f]
HB	DS_O3C3_	1 3 6	NoDup	2.9	117	"60 180 300"	
O[f]HC(Any)(Any)C(Any)(Any)Any							
HB	DS_N3C3_	1 4 5	NoDup	2.9	110	"60 180 300"	N[f]H2ZC{Z:C&!C=O&!C:Hev}
HB	DS_O2S_	3 2 1	All	2.9	120	"0.0 180"	AnyS(=O)(=O)NH
#TYPE	NAME	ATOMS	SEARCH	DIST	LP	ATYPE	Query
=====	=====	=====	=====	=====	=====	=====	=====
HBex	DS_O3C3_	2 1 3	NoDup	2.9	YES	"LP H"	
O[f]HC(Any)(Any)Z{Z:Hev&!C(Any)(Any)Any}							
HBex	DS_O3C3_	3 1 2	NoDup	2.9	YES	"LP"	O[f](Z)Z{Z:C&!C=Het}
HBex	DS_N3C3_	2 1 4	NoDup	2.9	""	"H"	
N[f]H2YaZ{Z:Hev&!C}{Ya:C&!C=O&!C:Hev}							
HBex	DS_N3C3_	2 1 3	NoDup	2.9	YES	"LP H"	N[f]H(Ya)Ya{Ya:C&!C=O&!C:Hev}
HBex	DS_N3C3_	3 1 2	NoDup	2.9	YES	"LP"	
N[f](Ya)(Ya)Ya{Ya:C&!C=O&!C:Hev}							
HBex	DS_N2C2_	2 1 3	NoDup	3.0	YES	"H LP"	N[f]H=C
HBex	DS_N2C2_	1 2 3	NoDup	3.0	YES	"H LP"	Any~N[f]=C
HBex	DS_N2C2_	1 2 3	NoDup	3.0	YES	"LP"	Any~N[r]=C[r]
HBex	DS_N2N2_	2 1 3	NoTriv	3.0	YES	"LP H"	N[1]H:C:C:N[f]:C:@1
HBex	DS_N2N2_	2 1 3	NoTriv	3.0	YES	"LP H"	N[1]H:C:C:N[f]:C:@1
HBex	DS_N2N2_	3 2 1	NoDup	3.0	YES	"LP"	C:N[f]:Hev
hb	DS_O3S_	3 2 1	NoDup	2.9	128	"0.0 180.0"	HevS=O[f]
hb	DS_O3S_	4 2 1	All	2.9	128	"0.0 180.0"	HevS(=O[f])=O[f]
hb	DS_O3S_	4 2 1	All	2.9	128	"0.0 180.0"	HevS(~O[f])(~O[f])~O[f]
hb	DS_O3N_	3 2 4	All	2.9	128	"0.0 180.0"	HevN(O[f])O[f]
hb	DS_O2N_	4 2 1	NoDup	2.9	128	"0.0 180.0"	HevN(Hev)~O[f]
hbex	DS_N2N2_	3 2 1	NoDup	3.0	YES	"LP"	N:N[f]:N
hb	DS_O3P_	3 1 2	All	2.9	128	"0.0 180.0"	P(~O)(~O)(~O)(~O)
hb	DS_O3P_	3 1 2	All	2.9	128	"0.0 180.0"	P(~O)(~O)(~O)
# CLASSNAMES#	Acceptor_site	Donor_Atom	DL				
HB	AS_HO3C2_	1 3 4	All	2.9	119	"0.0 180.0"	O[f]HC(:Hev):Hev
HB	AS_HO3C3_	1 3 6	NoDup	2.9	117	"60 180 300"	
O[f]HC(Any)(Any)C(Any)(Any)Any							
HB	AS_N3C3_	1 4 7	NoDup	2.9	110	"60 180 300"	
N[f]H2C(Any)(Any)C(Any)(Any)Any							
HB	AS_N3C3_	1 5 8	NoDup	2.9	110	"60 180 300"	
N[f]H3C(Any)(Any)C(Any)(Any)Any							
#TYPE	NAME	ATOMS	SEARCH	DIST	LP	ATYPE	Query
=====	=====	=====	=====	=====	=====	=====	=====

```

#==== ===== ===== ===== ===== ===== =====
HBex AS_HN2C2_ 2 1 3 NoDup 3.0 "" "H" NHC(Any)=O[f]
HBex AS_HN2C2_ 3 2 1 NoDup 3.0 YES "LP H" C:N[f]H:Hev
HBex AS_HN2C2_ 6 5 4 NoTriv 3.0 YES "LP" N[1]H:C:C:N[f]:C:@1
HBex AS_HO3C3_ 2 1 3 NoDup 2.9 YES "LP H"
O[f]HC(Any)(Any)Z{Z:Hev&!C(Any)(Any)Any}
HBex AS_HN2C2_ 3 2 4 Nodup 3.0 YES "LP H" HevN[f]H=C
HBex AS_HN2C2_ 1 2 3 Nodup 3.0 YES "LP" HevN[f]=C
HBex AS_HN2C2_ 2 1 4 Nodup 3.0 "" "H" N[f]H2C(N)=N
HBex AS_N3C3_ 2 1 4 Nodup 2.9 YES "LP H"
N[f]H2C(Any)(Any)Z{Z:Hev&!C(Any)(Any)Any}
HBex AS_N3C3_ 2 1 5 Nodup 2.9 YES "LP H"
N[f]H3C(Any)(Any)Z{Z:Hev&!C(Any)(Any)Any}
HBex AS_N3C3_ 2 1 3 NoDup 2.9 YES "LP H" N[f]H(Ya)Ya{Ya:C&!C=O&!C:Hev}
HBex AS_N3C3_ 2 1 4 NoDup 2.9 YES "LP H" N[f]H2(Ya)Ya{Ya:C&!C=O&!C:Hev}
HBex AS_N3C3_ 2 1 3 NoDup 2.9 YES "LP H" N[f]H(Ya)(Ya)Ya{Ya:C&!C=O&!C:Hev}
HBex AS_N3C3_ 3 1 2 NoDup 2.9 YES "LP" N[f](Ya)(Ya)Ya{Ya:C&!C=O&!C:Hev}
HBex AS_HN2C2_ 2 1 3 NoDup 3.0 YES "H LP" N[f]H=C
HBex AS_HN2C2_ 3 1 2 NoDup 3.0 YES "LP" N[f]=C~Any
HBex AS_HN2C2_ 2 1 4 NoDup 3.0 "" "H" N[f]H2Hев(:Hев):Hев
HBex AS_HN2C2_ 2 1 3 NoDup 3.0 "" "H" N[f]HHev(:Hев):Hев
HBex AS_HN2C2_ 1 2 3 NoDup 3.0 "" "H" HNC=Any
HBex AS_HNS3_ 6 5 2 NoDup 3.0 "" "H" AnyS(=O)(=O)N[f]H
HBex AS_HN4_ 2 1 3 NoDup -3.6 "" "C*" N[f](Z)(Z)(Z)Z{Z:C&!C=O&!C:Hev}
hbex AS_HN2N2_ 3 2 1 NoDup 3.0 YES "LP" N:N[f]:N
hb AS_O3P_ 3 1 2 All 2.9 128 "0.0 180.0" P(~O)(~O)(~O)(~O)
hb AS_O3P_ 3 1 2 All 2.9 128 "0.0 180.0" P(~O)(~O)(~O)

```

## APPENDIX "C"

### EXPERIMENTAL DATA SETS

<u>Data Set</u>	<u>No. Of Cpd's</u>	<u>Structure, Activity</u>
1 Uehling	9	camptothecin, DNA fragmentation
2 Strupczewski	34	benzisoxazoles, ip Behavioral
3 Siddiqi	10	adenosines, Brain A1 binding
4 Garratt1	10	tryptamines, melanophore binding
5 Garratt2	14	tryptamines, melanophore binding
6 Heyl	11	deltorphin, opioid receptor (DAMGO)
7 Cristalli	32	adenosines, A2a agonists
8 Stevenson	5	piperidines, NK1 antagonism
9 Doherty	6	triarylbutenolides, endothelin-A antag.
10 Penning	13	SC-41930 analogs, LTB4 antagonism
11 Lewis	7	oxazolinediones, NK1 binding
12 Krystek	30	sulfonamides, endothelin-A antagonism
13 Yokoyama1	13	oxamic acids, T3 binding
14 Yokoyama2	12	oxamic acids, T3 binding
15 Svensson	13	benzindoles, 5-HTA agonism
16 Tsutsumi	13	peptidyl heterocycles, endopeptidase inhib
17 Chang	34	biphenyl sulfonamides, AT1 binding
18 Rosowsky	10	trimetrexate analogs, DHFR inhibition
19 Thompson	8	peptidomimetic, HIV-1 protease inhibition
20 Depreux	26	naphthylethyl amides, melatonin displ.

### Literature References for Data Sets:

1. Uehling, D.E., Nanthakamur, S.S., Croom, D., Emerson, D.L., Leitner, P.P., Luzzio, M.J., *et al.*, Synthesis, Topoisomerase I Inhibitory Activity, and in Vivo Evaluation of 11-Azacamptothecin Analogs. *J. Med. Chem.* 1995, 38, 1106 (Table 2, with R<sub>2</sub>=Et; IC<sub>50</sub> data).
2. Strupczewski, J.T., Bordeau, K.J., Chiang, Y., Glamkowski, E.J., Conway, P.G., *et al.* 3-[[aryloxy)alkyl]piperidinyl]-1,2-Benzisoxazoles as D2/5-HT2 Antagonists with Potential Atypical Antipsychotic Activity: Antipsychotic Profile of Iloperidone

- (HP873). *J. Med. Chem.* 1995, 38, 1119. (Tables 2 and 3 with n=3, X=O; ED<sub>50</sub> for inhibition of apomorphine-induced climbing.)
3. Siddiqi, S.M., Jacobson, K.A., Esker, J.L., Olah, M.E., Ji, Xi.-duo., *et al.*, Search for New Purine- and Ribose-Modified Adenosine Analogs as Selective Agonists and Antagonists at Adenosine Receptors. *J. Med. Chem.* 1995, 38, 1174. (Table 1, R<sub>2</sub>=H; K<sub>1</sub>(A1), values estimated from % displacement and stereoisomers averaged as needed.)
4. Garratt, P. J., Jones, R., Tocher, D. A., Sugden, D., Mapping the Melatonin Receptor. 3. Design and Synthesis of Melatonin Agonists and Antagonists Derived from 2-Phenyltryptamines. *J. Med. Chem.* 1995, 38, 1132. (Table 1 and Table 2).
5. Garratt, P. J., Jones, R., Tocher, D. A., Sugden, D., Mapping the Melatonin Receptor. 3. Design and Synthesis of Melatonin Agonists and Antagonists Derived from 2-Phenyltryptamines. *J. Med. Chem.* 1995, 38, 1132. (Table 1 and Table 2).
6. Heyl, D.L., Dandabuthla, M., Kurtz, K.R., Mousigian, C. Opioid Receptor Binding Requirements for the &-Selective Peptide Deltorphin I: Phe<sup>3</sup> Replacement with Ring-Substituted and Heterocyclic Amino Acids. *J. Med. Chem.* 1995, 38, 1242. (Table 1; binding K<sub>1</sub> to DAMGO.)
7. Cristalli, G., Camaioni, E., Vittori, S., Volpini, R., Borea, P.A., *et al.* 2-Aralkynyl and 2-Heteroalkynyl Derivatives of Adenosine-5'-N-ethyluronamide as Selective A<sub>2a</sub> Adenosine Receptor Agonists. *J. Med. Chem.* 1995, 38, 1462.
8. Stevenson, G.I., MacLeod, A.M., Huscroft, I., Cascieri, M.A., Sadowski, S., Baker, R. 4,4-Disubstituted Piperidines: A New Class of NK<sub>1</sub> Antagonist. *J. Med.*

- Chem.* **1995**, *38*, 1264. (Table 1.)
9. Doherty, A.M., Patt, W.C., Edmunds, J.J. Berryman, K.A., Reisdorph, B.R., *et al.* Discovery of a Novel Series of Orally Active Non-Peptide Endothelin-A (ET<sub>A</sub>) Receptor-Selective Antagonists. *J. Med. Chem.* **1995**, *38*, 1259. (Table 3; IC<sub>50</sub> ET<sub>A</sub>.)
10. Penning, T.D., Djuric, S.W., Miyashiro, J.M., Yu, S., Snyder, J.P., *et al.* Second-Generation Leukotriene B<sub>4</sub> Receptor Antagonists Related to SC-41930; Heterocyclic Replacement of the Methyl Ketone Pharmacophore. *J. Med. Chem.* **1995**, *38*, 858. (Table 1, all; LTB<sub>4</sub> receptor binding.)
11. Lewis, R.T., MacLeod, A.M., Merchant, K.J. Kelleher, F., Sanderson, I., *et al.* Tryptophan-Derived NK1 Antagonists: Conformationally Constrained Heterocyclic Bioisosteres of the Ester Linkage. *J. Med. Chem.* **1995**, *28*, 923.
12. Krystek, S.R., Hunt, J.T., Stein, P.D., Stouch, T.R. 3D-QSAR of Sulfonamide Endothelin Inhibitors. *J. Med. Chem.* **1995**, *38*, 659.
13. Yokoyama, N., Walker, G.N., Main, A.J. Stanton, J.L. Morrissey, M., *et al.* Synthesis and SAR of Oxamic Acid and Acetic Acid Derivatives Related to L-Thyronine. *J. Med. Chem.* **1995**, *38*, 695.
14. Yokoyama, N., Walker, G.N., Main, A.J. Stanton, J.L. Morrissey, M., *et al.* Synthesis and SAR of Oxamic Acid and Acetic Acid Derivatives Related to L-Thyronine. *J. Med. Chem.* **1995**, *38*, 695.
15. Haadsma-Svensson, S.R., Svensson, K., Duncan, N., Smith, M.W., Lin, Ch.-H. C-9 and N-Substituted Analogs of cis-(3aR)-(-)-2,3,3a,4,5,9b-Hexahydro-3-propyl-1H-benz[e]indole-9-carboxamide: 5HT1A Receptor Agonists with Various Degrees of

- Metabolic Stability. *J. Med. Chem.* 1995, 38, 725.
16. Tsutsumi, S., Okonogi, T. Shibahara, S., Ohuchi, S., Hatushiba, E., *et al.*, Synthesis and Structure Activity Relationships of Peptidyl  $\alpha$ -Keto Heterocycles as Novel Inhibitors of Prolyl Endopeptidase. *J. Med. Chem.* 1994, 37, 3492. (Table 2, X=CH<sub>2</sub>CH<sub>2</sub>; IC<sub>50</sub>.)
17. Chang, L.L., Ashton, W.T., Flanagan, K.L., Chen, Ts.-Bau., O'Malley, S.S., *et al.*, Triazolinone Biphenylsulfonamides as Angiotensin II Receptor Antagonists with High Affinity for Both the AT<sub>1</sub> and AT<sub>2</sub> Subtypes. *J. Med. Chem.*, 1994, 37, 4464. (Table 1, R<sup>3</sup> =(2-C1)C<sub>6</sub>H<sub>5</sub>; AT<sub>1</sub> [rabbit aorta] IC<sub>50</sub>.)
18. Rosowsky, A., Mota, C.E., Wright, J.E., Queener, S.F., 2,4-Diamino-5-chloroquinazoline Analogs of Trimetrexate and Piritrexim: Synthesis and Antifolate Activity. *J. Med. Chem.* 1994, 37, 4522. (Table 2; rat liver IC<sub>50</sub>.)
19. Thompson, S.K., Murthy, K.H.M., Zhao, B., Winborne, E., Green, D.W., *et al.* Rational Design, Synthesis, and Crystallographic Analysis of a Hydroxyethylene-Based HIV-1 Protease Inhibitor Containing a Heterocyclic P1'-P2' Amide Bond Isostere. *J. Med. Chem.* 1994, 37, 3100. (Table 2, X-Boc; apparent K<sub>i</sub>.)
20. Depreux, P., Lesieur, D., Mansour, H.A., Morgan, P., *et al.* Synthesis and Structure-Activity Relationships of Novel Naphthalenic and Bioisosteric Related Amidic Derivatives as Melatonin Receptor Ligands. *J. Med. Chem.* 1994, 37, 3231.

## APPENDIX "D"

A list of 736 commercially available thiols broken down into 231 clusters based on topomeric CoMFA field descriptors along with the systematic name applicable to each. The 231 clusters are sorted by proposed name, first by the "root" structure, ie., the fragment attached immediately to the -SH, and then by the substitution pattern on that "root" substructure. The names describe topologically equivalent hydrocarbons, ie., structures in which all monovalent atoms are replaced by hydrogens and the other atoms by carbons.

Cluster ID	Cluster Size	Struct. Root	Structural Substitution <sup>a</sup>
1	26	aryl	Simple
144	1	aryl	2,3,5-Me
177	1	aryl	2,3,5-Me-4-Pr
163 <sup>c</sup>	1	aryl	2,3-(4-(2,3-Pr)5het)5hetO
151	1	aryl	2,3-(4-Bu)5hetO-5-Me
33	5	aryl	2,3-Benzo
80	2	aryl	2,5-Me
192	1	aryl	2,5-Me-3-iPe
7	14	aryl	2,6-NoH-3(4/5)-Me
27	6	aryl	2,6-NoH-3-Ar
107	2	aryl	2-(2-Bz)PheEt-4,5-Benzo
189	1	aryl	2-(3,5-Me)Ar-4,5-Benzo
141	1	aryl	2-(4-Et)PhePr
205	1	aryl	2-(4-Stilbenyl)Stilbenyl
188	1	aryl	2-5hetCH2-4,5-Benzo
56	3	aryl	2-Ar
138	1	aryl	2-Ar-3,5-Me
190	1	aryl	2-Ar-4,5-(3,4-Et)Benzo
41	6	aryl	2-Ar-4,5-Benzo
152	1	aryl	2-Bz
16	9	aryl	2-Et
85	2	aryl	2-NoH-3-Et-5-Me
106	2	aryl	2-PheEt-4,5-Benzo
77	2	aryl	2-PhePr
142	1	aryl	2-R8
121	2	aryl	2-Stilbenyl
97	2	aryl	3,4-(3-Me)Benzo
218	1	aryl	3,4-(a,b)IndenO
164	1	aryl	3,4-(a,b,(8-Ar)IndenO)-6-Me
98	2	aryl	3,4-(a,b,(c-Me)IndenO)
99	3	aryl	3,4-(a,b-Naphtho)
157	1	aryl	3,4-Ar
58	3	aryl	3,4-Benzo-5-Me
100	2	aryl	3,4-Benzo-6-tBu
37	5	aryl	3,5-Me
180	1	aryl	3-(2,3-Benzo-4-Et)5het
199	1	aryl	3-(2,3-Benzo-5-Me)5het
182	1	aryl	3-(2-Me-3-5het-5-Et)5het
115	2	aryl	3-(3-5het)5het
193	1	aryl	3-(3-Ar)5het-4-Me
67	3	aryl	3-Ar
129	2	aryl	3-Ar-4-(2-Me)5hetCH2
46	4	aryl	3-Ar-5-Me
155	1	aryl	3-Bz
82	2	aryl	3-Bz-5,6-Benzo
10	16	aryl	3-Me

70	3	aryl	3-Naphth
73	3	aryl	3-Pr-4-sBu-6-Me
95	2	aryl	3-iPr
88	2	aryl	4-Ar
81	2	aryl	4-Bz
48	4	aryl	4-Et
2	23	aryl	4-Me
92	2	aryl	4-R9+
90	4	aryl	4-iBu
19	8	aryl	6-NoH
148 <sup>c</sup>	1	aryl	(adenosine)
228	1	aryl	(fluorescein)
12	10	5het	Simple
50	4	5het	2,3-(a,b-Naphtho)
139	1	5het	2,3-5hetO-4-Me
89	2	5het	'2,3-Ar
173	1	5het	2-(2,5-Et)Ar-3-Et
69	3	5het	2-(2-Me)Ar-3-(2-Me)PheEt
198	1	5het	2-(2-Me)Ar-3-R10
174	1	5het	2-(2-sBu)-3-Et
171	1	5het	2-(3,5-Me)Ar-3-5het
170	1	5het	2-(3,5-Me)Bz-3,4-Benzo
123	2	5het	2-(3-Et)Ar-3-Bz
22	7	5het	2-(4-Et)Ar
202	1	5het	2-(4-Et)Ar-4-(4-Me)Ar
122	2	5het	2-(4-iPr)Ar-3-Bz
197	1	5het	2-5hetCH2-3-(4-tBu)Ar
6	14	5het	2-Ar
225	1	5het	2-Ar-3-(2-Ar)5hetBu
224	1	5het	2-Ar-3-(2-Ar)5hetCH2
63	3	5het	2-Ar-3-(2-Bz)Ar
178	2	5het	2-Ar-3-(2-Me)5het
72	3	5het	2-Ar-3-(3,4-Et)Bz
40	5	5het	2-Ar-3-(3-Ar)5HetEt
183	1	5het	2-Ar-3-(3-Ar)PhePr
64	3	5het	2-Ar-3-(3-Ar-5-Me)5het
105	2	5het	2-Ar-3-(3-Me)Ar
160	1	5het	2-Ar-3-(4-Ar)Cyhx
146	1	5het	2-Ar-3-(4-Ar)CyhxCH2
203	1	5het	2-Ar-3-(4-PheEt)Ar
126	2	5het	2-Ar-3-(tBu)Ar
17	9	5het	2-Ar-3-Ar
211 <sup>c</sup>	1	5het	2-Ar-3-Benzylidene
124	2	5het	2-Ar-3-IndenCH2
28 <sup>b</sup>	6	5het	2-Ar-3-Me
30	6	5het	2-Ar-3-PhePr
204	1	5het	2-Ar-5-(4-(2,4-Me)Bz)Ar
79	2	5het	2-Bz
78	2	5het	2-Bz-3,4-Benzo
117	2	5het	2-Cyhx
186	1	5het	2-Cyhx-3,4-iPe
68	3	5het	2-Et
112	2	5het	2-Et-3-(2-Me)PheEt

128	2	5het	2-Me-3,4-(3-Me) Benzo
93	2	5het	2-Me-3,4-Benzo
61	3	5het	2-Me-3-(2,3,4-Me) 5het
181	1	5het	2-Me-3-(2,3-Benzo-4-Et) 5het
49	4	5het	2-Me-3-(3-Ar) 5het
86	2	5het	2-Me-3-(3-Ar) 5hetPr
91	2	5het	2-Me-3-(3-Ar-5-Me) 5het
4	17	5het	2-Me-3-(3-Bz) Ar
172	1	5het	2-Me-3-(4-tBu) PheEt
38	5	5het	2-Me-3-5HET
13	10	5het	2-Me-3-Me
222	1	5het	2-Me-3-Pe
66	3	5het	2-Me-3-PheEt
29	6	5het	2-Me-3-PhePr
71	3	5het	2-Me-3-R8+
108	2	5het	2-Me-5-Bu
127	2	5het	2-Pe-3-Ar
54	3	5het	2-Pr
221	1	5het	2-R12
187	1	5het	2-iBu-3,4-iPe
143	1	5het	2-iPe-3,4-Benzo
96	2	5het	3,4-(2,4-Me) Benzo
162	1	5het	3,4-(3-Ar) Benzo
169	1	5het	3,4-(3-Hx) Benzo
94	2	5het	3,4-(3-Pr) Benzo
210	1	5het	3,4-(a,b-Napththo)
36	15	5het	3,4-Benzo
176	1	5het	3-(2,4-Me) Bz
196	1	5het	3-(3,5-Me) Ar
159	1	5het	3-(3-Ar) 5het
42	4	5het	3-(3-Bz) Ar
200	1	5het	3-(3-Me) PheEt
113	2	5het	3-(4-Me) Ar
125	2	5het	3-(4-tBu) Ar
191	1	5het	3-(A1-4-Et) PheEt
145	1	5het	3-(B-Ar) PhePr
114	2	5het	3-5hetCH2
18	8	5het	3-Ar
59	3	5het	3-Ar(2-thia)
65	3	5het	3-Bu
24	7	5het	3-Me-5-H
44	6	5het	3-Me-5-NoH
52	5	5het	3-Pe
111	2	5het	3-PheEt
153	1	5het	3-PhePr
32 <sup>b</sup>	6	5het	3-Pr
223	1	5het	3-R13
185	1	5het	(chrysenO)
34	5	alkyl	Simple
104	2	alkyl	(3)(B1)(B1)
62	3	alkyl	(3-Me) PhePr
3	18	alkyl	(3:4)
14	9	alkyl	(3:4)(A1)

60	3	alkyl	(3:4) (B1)
226	1	alkyl	(4) (A1) (A-tBu) (C1) (C1)
45	4	alkyl	(4) (D1) (D1)
35	7	alkyl	(4-Me) PhePr
168	1	alkyl	(4-iPe) PhePr
47	4	alkyl	(5) (A1)
179	1	alkyl	(5) (B1) (E-(2-Ar-5-Me) 5het)
103	2	alkyl	(5) (B3)
76	2	alkyl	(5) (C1) (C1)
83	2	alkyl	(5) (C2)
216	1	alkyl	(5) (C2) (D2) (D2)
43	8	alkyl	(5:6) (D1/B1/F1)
5	15	alkyl	(5:7)
158	1	alkyl	(6) (B8) (C1) (E1) (E1)
140	1	alkyl	(6) (F-Ar)
166	1	alkyl	(7) (A8) (F1)
53	3	alkyl	(7) (D3) (D3)
207	1	alkyl	(8) (C3)
8	13	alkyl	(8:11)
206	1	alkyl	(9) (B4) (G3)
75	3	alkyl	(10) (B1) (E5) (E1)
136	1	alkyl	(10) (C1) (E5) (E2)
20	8	alkyl	(10+) (B1)
39	7	alkyl	(11+) (B1)
154 <sup>c</sup>	1	alkyl	(12) (A-PheEt)
230	1	alkyl	(12) (F6) (F1)
131	2	alkyl	(12) (F6) (F6)
15	9	alkyl	(12+)
137	1	alkyl	(13) (E4)
231	1	alkyl	(A-Ar) (A-Ar) Bz
229	1	alkyl	(A-Bz) (A-Bz) PheEt
184	1	alkyl	(A1) PheEt
227 <sup>c</sup>	1	alkyl	(cholesterol)
214 <sup>c</sup>	1	alkyl	(cryptate)
23	7	alkyl	PheBu
74	3	alkyl	PheEt
25 <sup>b</sup>	6	alkyl	PhePr
11	10	benzyl	Simple
102	2	benzyl	2,4,5-Me
57	3	benzyl	2,4,6-Me
217	2	benzyl	2-(3-(2-Et)Ar)Ar
213	1	benzyl	2-Et-3-(2,3-Et-5-Me)Ar-5-Me
212	1	benzyl	2-R8-3-Naphthyl-4,5-Benzo
9	13	benzyl	2/3-Me
84	2	benzyl	3,4-Benzo
132	2	benzyl	3,5-Me
130	2	benzyl	3-(4-Stilbenyl)Stilbenyl
134	2	benzyl	4-(3-Ar)Ar
21	7	benzyl	4-Et
26 <sup>b</sup>	6	benzyl	4-Me
156	1	benzyl	4-PhePr
201	1	benzyl	4-tBu
135	2	alkenyl	Ar..(2-Et)Ar

220	1	alkenyl	Ar..(4-Bz)Ar
116	2	alkenyl	Ar..Ar
133	2	alkenyl	Ar..Bz
110	2	alkenyl	Et.CN.CONH2
87	2	alkenyl	NH2.CN.N=NPh
119	2	alkenyl	P(NMe2)3..Ar
120	2	alkenyl	P(Pr)3..Ar
118	2	alkenyl	P(iPe)3..Ar
51	4	alkenyl	PCyhx3..Ar
195 <sup>c</sup>	1	alkenyl	PEt3..(2-Bz)Ar
31 <sup>b</sup>	6	alkenyl	PEt3..Ar
194	1	alkenyl	PEt3..Bz
109	2	alkenyl	PheEt.CN.CONH2
101	2	cyclohexyl	Simple
149	1	cyclohexyl	1-Me-2,4-CMe2
55	3	cyclohexyl	2,3,4,5-iBu
147	1	cyclohexyl	2,3,4-iBu-5-iPe
209	1	cyclohexyl	2-(3,4-PheEt)5het-6-Me
208	1	cyclohexyl	2-Me-3,5-CMe2
167	1	cyclohexyl	2-Me-4-sPe
165	1	cyclohexyl	2-iPr-3,5-Me
150	1	cyclohexyl	3-sPe-6-Me
161	1	cyclohexyl	4-Et-4-iBu
219	1	cyclohexyl	(complex)
175	1	cyclopentyl	2-Ar-4-spiro
215	1	cyclopentyl	3-PhePr

<sup>a</sup>To generate these names, all heteroatoms are first replaced by carbon (to produce the simplest common topology) and a particular structure is chosen from among these topologies as the "most typical" of that cluster, if possible to contain the largest substructure that distinguishes that cluster from all others.

Within the name of a substitution, numbers indicate positions when substitution is on a ring, but chain length when substitution is on a chain (numbers separated by a colon indicate a range of chain lengths). Also, within a chain, letters indicate a position of substitution. (For example, (C2) describes a two atom branching from the third position of a chain, while 3-PhePr describes a phenyl propyl skeleton attached to the 3-position of a ring. )

A dot notation (.) separates the three possible substituents on an alkenyl root, the substituent order being same carbon as the -SH substituent, then the position *trans* to the -SH, and finally *cis* to -SH.

The above notwithstanding, any name enclosed completely in parentheses takes its usual structural meaning.

Here are structural descriptions for each name abbreviation in the above table, mostly in SLN (SYBYL Line Notation), listed alphabetically. (SLN extends SMILES with the following concepts, among others. Hydrogens are explicit. Ring openings and closures begin with a number enclosed by [] and end with the matching number preceded by @. Other SLN symbols used in these SLN definitions are: ~ = any bond; - = single bond (used here to provide a reference for [R]) : = aromatic bond; ! = the SLN following (here in parentheses) is not allowed; [F] = no additional atoms may be attached to the preceding atom; [!R] = preceding bond may not be in a ring; [R] = preceding bond must be in a ring.)

**5Het** = **5H**et = C[1]:C:C:C:C:@1. **alkenyl** = C=C. **aryl** = C-[!R]C. **aryl** = C-[!R]C. **benzyl** = Bz = HSC-[!R]C~[R]C. **Bu** = C-[!R]C-[!R]C-[!R]C-[!R]C. **cyclohexyl** = Cyhx = C[1](-!-)C~C~C~C~C~@1. **cyclopentyl** = C[1]~(-!-)C~C~C~C~@1. **Et** = C-[!R]C. **inden** = C[1]:C(~C~X~[2]):C(~@2):C:C@1. **iBu** = C-[!R]C-[!R]C(-[!R]C)-[!R]C. **iPe** = C-[!R]C-[!R]C-[!R]C(-[!R]C)-[!R]C. **Me** = C. **naphth** = C[1]:C(~C~X~[2]):C(~@2):C:C:C@1. **NoH** = !(CH). O denotes ring fusion, e.g., **benzo** fuses a 6-membered aromatic ring. **Pe** = C-[!R]C-[!R]C-[!R]C-[!R]C-[!R]C. **Pr** = C-[!R]C-[!R]C-[!R]C. **R#** = alkyl chain of approximate length #. **Simple** = !(C~[!R]C). **sPe** = C(-[!R]C)-[!R]C-[!R]C-[!R]C-[!R]C. **Stilbenyl** = C=[!R]C-[!R]C[1]:C:C:C:C@1. **tBu** = C(-[!R]C)(-[!R]C)-[!R]C.